

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction



Master's Thesis

3D interior design software for tablet computers

Peter Šomló

Supervisor: Ing. Michal Hapala

Study Programme: Open Informatics

Study Specialization: Computer Graphics and Interaction

May 2013

Declaration – Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v příloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu

§ 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 9.5.2013

Peter Šomló

Abstract

This thesis deals with creating an interior design application for tablet computers. It begins with the assessment of currently available interior design applications for both desktops and tablets. It discusses the possibilities of building such application on the iPad. The specific steps and techniques needed to build the application are described. Suggestions for further development based on user feedback are proposed.

Abstrakt

Diplomová práce se zabývá problematikou aplikací pro interiérový design. Analyzuje schopnosti dostupných nástrojů pro modelování interiérů pro desktopy a tablety. Navrhuje postup pro implementaci aplikace pro tablet iPad. Popisuje a rozebírá kroky k vytvoření dané aplikace. Na základě zpětné vazby uživatelů navrhuje možnosti rozšíření.

Contents

1	Introduction	12
1.1	Statement of Problem	12
2	Background Survey	13
2.1	Overview.....	13
2.1.1	Technology Platform.....	13
2.1.2	Target Users	13
2.2	Desktop Applications Overview.....	14
2.2.1	IKEA Kitchen Planner.....	14
2.2.2	Sweet Home 3D	15
2.3	iPad Application Overview	17
2.3.1	Home Design 3D	18
2.3.2	Home 3D	21
2.3.3	Living Room for iPad.....	24
2.4	Conclusion.....	26
3	Platform Description	28
3.1	The iPad.....	28
3.2	Objective-C and Cocoa Touch	28
3.3	Gestures.....	30
3.4	Conclusion.....	32
4	Design and implementation.....	32
4.1	Scene graph – iSGL3D.....	33
4.2	Menu	34
4.2.1	Storyboards	34
4.2.2	Target-action mechanism.....	36
4.3	Gestures.....	37
4.3.1	Designing the gestures.....	37
4.3.2	UIGestureRecognizer.....	38
4.3.3	Touch events in iSGL3D	39
4.3.4	Pan to move	40
4.3.5	Pinch to zoom	41
4.3.6	Rotation.....	42
4.3.7	Combining gestures	43

4.3.8	Conclusion.....	44
4.4	Walls	44
4.4.1	The user interface.....	45
4.4.2	Winged-edge data structure	46
4.4.3	Enumerating the winged-edge structure.....	48
4.4.4	Geometry	48
4.4.5	Drawing walls	49
4.4.6	Removing walls and delegation	50
4.4.7	Creating rooms	52
4.4.8	Blocks.....	54
4.5	Furniture.....	54
4.5.1	Navigating the furniture library.....	55
4.5.2	Table views	55
4.5.3	Blocks for threads/loading models	56
4.6	Mathematical functions and unit testing.....	57
5	Evaluation.....	59
5.1	Research Goals	59
5.2	Task Analysis/Usability Test	59
5.3	Findings.....	60
5.4	Design suggestions	60
6	Similarity search.....	61
6.1	The similarity search problem	61
6.1.1	Shape retrieval/categorization overview	61
6.2	Implementation.....	65
6.2.1	File download/scraping.....	66
6.2.2	Preview generation and file format transformation	67
6.2.3	Manual furniture selection	68
6.2.4	Text based classification.....	69
6.2.5	Geometric descriptor.....	74
6.2.6	Possible improvements and conclusion	78
7	Conclusion	80
8	Bibliography/references	82
9	Application screenshots.....	84
10	CD content	92

1 Introduction

One of the distinguishing parts of the process that leads to good design, be it software, product or interior design, is redesign. It could be described as the act of throwing away a piece of existing work and replacing it with a more suitable one [1]. This level of freedom in interior design can be achieved by either sketching an interior with a pencil or by using computers and software tools like CAD systems and specialized interior design software.

These software products enhance the ability to manipulate a design as compared to sketching on a paper. They stretch the possibilities of experimenting in 3D space. The improvement in making higher quality design iterations cheaper and easier has similar benefits as sketching on a paper has to moving physical objects around.

The real drawback of these advanced tools is the time and effort needed to learn and master them. To get enough benefit from them, one often has to spend weeks learning to use the tools efficiently. This often makes the tools inaccessible to hobby and amateur users.

One of the main reasons of this state is the discrepancy between natural understanding of objects in 3D space and their representation on a 2D screen. The discrepancy is further amplified by adding another level of separation – the mouse. The possibility to solve the first part of the problem, the 3D to 2D mapping, is still open after decades of research. The second part is a shortcoming of the desktop computing paradigm. This is beginning to be successfully removed by the advent of commercially available multitouch tablet computers that became widely adopted after the 2010 introduction of the Apple iPad.

A platform, which brings the real and the virtual worlds closer together, is worth investigating, as removing the gap between the two worlds makes computers even more pervasive and opens the possibility to accomplish a new range of tasks that can be solved better and more efficiently.

1.1 Statement of Problem

The problem this thesis aims at is making interior design applications more available to hobby and non-professional users by exploiting the multitouch navigation capabilities of tablets. Its goal is to deeper understand the problem by examining the current solutions (chapter 2 – Background Survey) examine the possibilities of the platform, suggest and implement a viable solution and test the basic assumptions.

2 Background Survey

2.1 Overview

Software applications can be divided into categories based on different criteria. The platform where the application runs and the usage scenario/target user group of the application are the two main categories.

2.1.1 Technology Platform

In computing, the term technology platform has a broad meaning ranging from hardware architecture (x86, ARM...) to software platforms like operating system, software frameworks, or APIs on the Internet.

For the purpose of this work, a "platform" is a system that can be programmed and therefore customized by outside developers – users – and in that way adapted to countless needs and niches that the platform's original developers could not have possibly contemplated [2].

The two most significant platform changes in the last years are the advent of the Internet and tablets. Internet, at its core, has a huge influence on the way data can be acquired and distributed, regardless of whether the end user is interacting with a native application or an application in the web browser.

Both native and web interior design applications benefit from the possibilities brought about by the Internet; sharing and cooperating on scenes, downloading additional models and software updates.

Besides the Internet, the highest potential to influence and improve the way people interact with interior design applications, as compared to the currently dominant ways of use, comes from tablet.

2.1.2 Target Users

Another way to divide the currently available applications is based on their target audience.

Professional CAD applications for architecture, engineering and construction were first introduced by companies like Graphisoft (ArchiCAD) and Autodesk (AutoCAD) in the 1980s, when desktop personal computers became capable of running such demanding products. Some vendors go even further and offer complex BIM systems that integrate software tools needed from the earliest stages of building design through its construction and operational life.

There are large manufacturing companies using AutoCAD and similar systems, but there are also other specialized design programs for kitchen and bathroom layouts, landscaping plans, and other homeowner-type situations.

One of the leaders in the field of interior-design software for residential and commercial markets is 20-20 Technologies. Its products are designed to work across all environments,

desktop to web; they provide various views (top-down, architectural, front-view) to generate a more realistic overview of the design for the client [3].

On the professional level, the applications can be further divided into the category, where the output is used for construction work and the category of applications (general purpose modelers like Google Sketchup or Autodesk Maya) used for generating images and 3D animations for visualization purposes.

Another target group is interior design professionals with applications like Punch! Interior Design, Punch! Home Design Studio, or Live Interior 3D. These tools are also often used by hobbyists. The price range is usually from tens to hundreds of euros.

2.2 Desktop Applications Overview

The main focus of the desktop applications overview is user interaction, browsing the object library, and methods of obtaining new models.

2.2.1 IKEA Kitchen Planner

Price: Free

Platforms: Windows, Mac OS

IKEA Home Planner (Figure 2-1) is a web-based tool for kitchen design. It requires a web browser plugin from 2020 Technologies to run. It offers means to draw rooms from scratch. A few popular kitchen setups are available through predesigned templates.

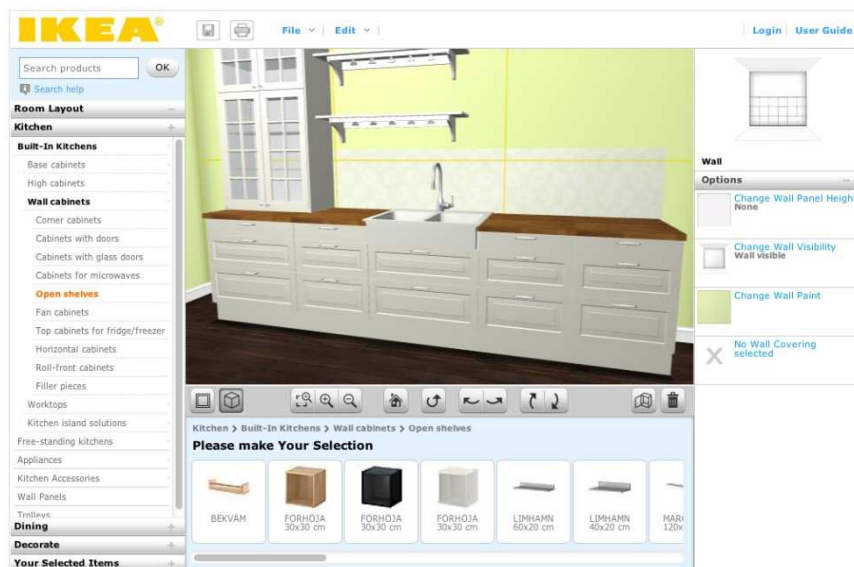


Figure 2-1

2.2.1.1 Navigation in scene



Figure 2-2

The program supports two viewing modes, 2D (floor view) and 3D (front view). The 3D mode supports camera rotation, tilt and zoom, which can be adjusted with the icons under the main pane (Figure 2-2). Panning is done by using the arrow keys.

2.2.1.2 Item library



Figure 2-3

The item library is limited to kitchen and dining furniture items from the IKEA store and a few basic appliances. Products are divided into categories, which are represented as a tree structure. Each subcategory contains around 10 furniture items. The items are displayed in the Item selector area at the bottom of the page (Figure 2-3).

Keyword based search is also available. It can handle article numbers, product names, measurements, colors and furniture functions.

2.2.2 Sweet Home 3D

Price: free/opensource

Platform: Java

Output and sharing: Printing, .obj model, .sh3d files, rendered images, video tour

The main window of the application (Figure 2-4) is divided into 4 panes, with a toolbar at the top.

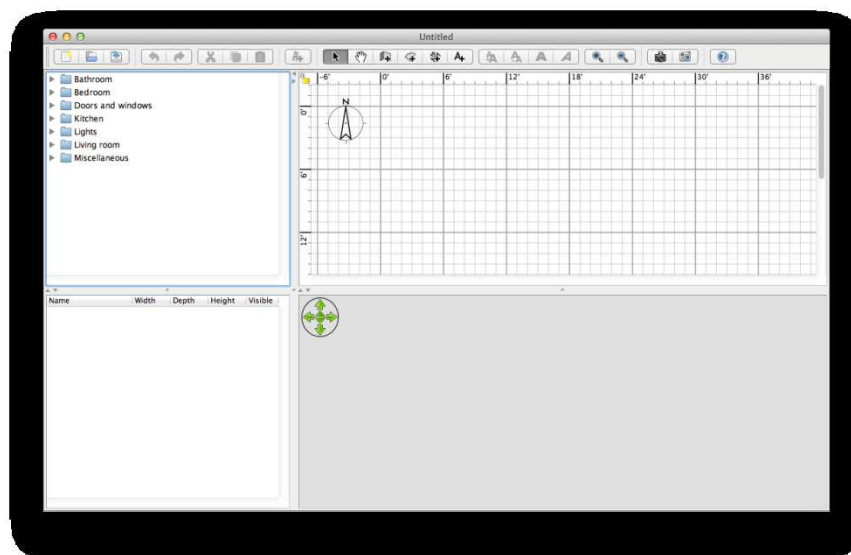


Figure 2-4

2.2.2.1 Manipulating items

Objects from the furniture library can be dragged to the 2D plan area, which can display a grid and rulers. Furniture models are automatically snapped to walls. The resulting scene is visualized in a 3D view in the bottom part of the main window. Objects can not be moved in the 3D view.

Object properties (name, dimensions...) are displayed in an editable table. Properties like wall colors and textures can be edited in the 2D view after selecting the object and right clicking the desired object.

2.2.2.2 Item library

Furniture models are packaged and distributed as model library files. They can be obtained from online sources and imported into the program. The supported model formats are OBJ, DAE, 3DS and LWS; therefore, various online model repositories (Google 3D Warehouse) can be used.

The application places the loaded models into a tree structure based on the room in which they are usually placed (Figure 2-5).

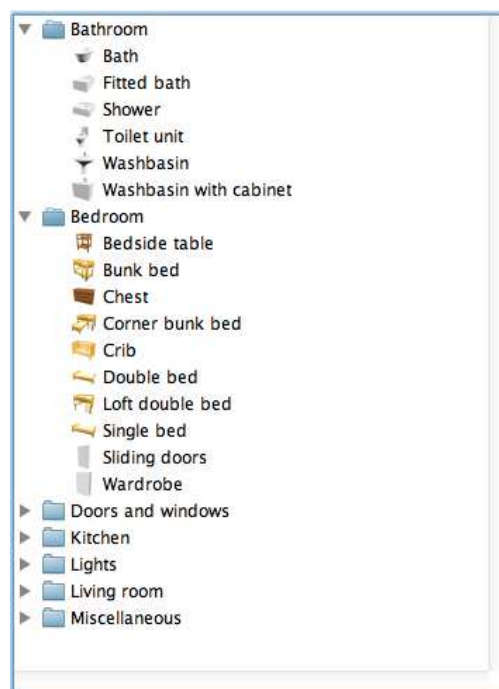


Figure 2-5

2.2.2.3 Disadvantages

- Scene is not editable in 3D mode, causing confusion between which view (top vs. bottom) to alter
- Weaker object placement constraints
- Need to switch cursor type/tool for selecting objects, panning and adding objects
- Inability to pan in 3D view (the camera position is always centered, problem with finding a good view for bigger models)

- Inability to group objects (to move a room every single wall has to be selected), planned for future releases
- Inability to rotate/scale multiple selected objects (both furniture, walls and floors) at once

2.2.2.4 Advantages

- Price
- Raising walls and floors can be done very precisely in 2D mode
- Ability to measure and set object sizes
- Multiple supported object formats with textures for import/export
- Ability to work on multistory buildings
- Display scanned blueprints in the background
- Focus on core functionality
- Plugin support

2.3 iPad Application Overview

There are two main categories of interior design applications. The first category supports 3D visualizations of the scene (Table 2-1). The 3D mode of these applications is at least partially interactive, i.e. some parts of the scene can be altered in these views. The second category is 2D applications without any means of 3D output (Table 2-2). The last tested category (Table 2-3) is for applications related to interior design focused on specific niches of interior design such as measurement, or choosing fabrics and materials for rooms.

This chapter introduces several terms specific for tablets used for describing the means of interaction. These are the specific patterns for finger motion called gestures, which are the input of the system. A more detailed discussion of the pan, zoom, pinch, and other gestures can be found in chapter 3.3.

iPad applications with 3D capabilities

Name	Price	Tested	Homepage	App Store	Notes
Home Design 3D	free/ 4.99 €	Free	www.livecad.net/EN/	free: 4/5 stars pro: 3.5/5 stars	in-app purchase
Home 3D for iPad	6.99 €	Yes	www.home3dapp.com	3/5 stars	in-app purchase
Room Design for iPad	5.99 €	Yes	www.roomdesignapp.com	2/5 stars	

Table 2-1

iPad applications with 2D capabilities

Name	Price	Tested	Homepage	App Store	Notes
Living Room for iPad	3.99 €	Yes	http://planetnextsoftware.com/livingroom/	3/5 stars	
Room Planner for iPad	6.99 €	No	http://www.blast-hd.co.jp/app/roomplanner		
Floorplans	19.99 €	No	http://www.floorplansapp.com/		

Home Space Planning Design Tool	1.59 €	No			
---------------------------------	--------	----	--	--	--

Table 2-2

Related iPad applications

Name	Price	Tested	Homepage	Notes
Photo Measures	free/3.99 €	Free		
uDecore	free/0.79 €	Free		Augmented Reality
ARange your room 3D	0.79 €	No		Augmented Reality
iProK3D Pro	free	Free		Viewer
3D room Room Creator Udesignit	4.99 €	No	http://udesignit.ca/	no furniture, but includes items from retailers

Table 2-3

2.3.1 Home Design 3D

The difference between the paid version and the free version, which has been tested, is in the ability to save scenes. Both versions allow for buying additional furniture models through in-app purchases.

2.3.1.1 Navigation in scene

The 2D navigation mode in Home Design 3D is similar to other widely used iPad applications with 2D navigation interfaces (the built-in maps, etc.). The pan gesture moves the scene around; pinch is used to zoom in and out. The scene can not be rotated in the 2D mode.

The 3D mode is used for scene exploration and walkthrough (with a fixed camera height). Objects can not be selected and interacted with. Panning with one finger rotates the camera view, in a trackball-like manner around the z-axis, which is perpendicular to the floor. The position of the z-axis is fixed to the scene origin.

The camera can be moved around with 4 arrow buttons in the lower left corner of the screen (Figure 2-6). The arrows move the camera in the respective direction under the current angle. However, the movement with the arrow keys and trackball rotation around scene is cumbersome. The two can not be used simultaneously without causing confusion as the center of the rotation is not the position of the camera, even though the arrows alter this position. The other gesture available in the 3D mode is the pinch gesture, which zooms in the scene. The rotation gesture is not used.

This approach to navigation is good for moving around the scene in a walkthrough mode, which feels like a first-person computer game. The most comfortable way of navigating is holding the tablet with both hands and placing both thumbs in the lower left and right corners of the screen, and using them to adjust to position as if they were placed on a joystick or a trackball.

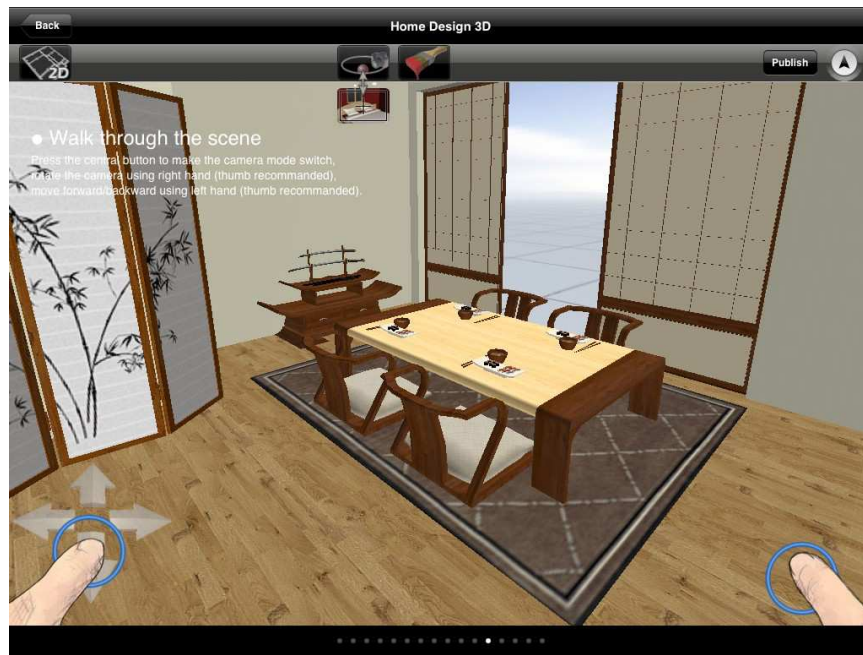


Figure 2-6

2.3.1.2 Manipulating items

The 2D mode is used for drawing and editing rooms. In order to draw a room in the top view, the user has to select the wall drawing tool. In this mode, the user draws a resizing rectangle by panning a finger. All rooms are rectangular. However the wall joins/vertices can be moved to skew the room. To create a room with more than 4 walls, one has to draw another rectangular room and set the joining wall to be invisible. The program still registers the rooms as two separate areas, but the visual effect looks as desired.

Tapping on a room opens a context menu with tools to alter the room - move, edit walls, edit corners, etc. (Figure 2-7). In the same fashion, a context menu is displayed after selecting a wall and furniture models.

Furniture models can be added in the 2D mode from a library. The models can be rotated, moved and scaled with the rotate, pinch and pan gesture, resulting in intuitive direct handling. No constraints are enforced on the objects; they can be placed over each other. Collision detection is not available. (Figure 2-8)



Figure 2-7

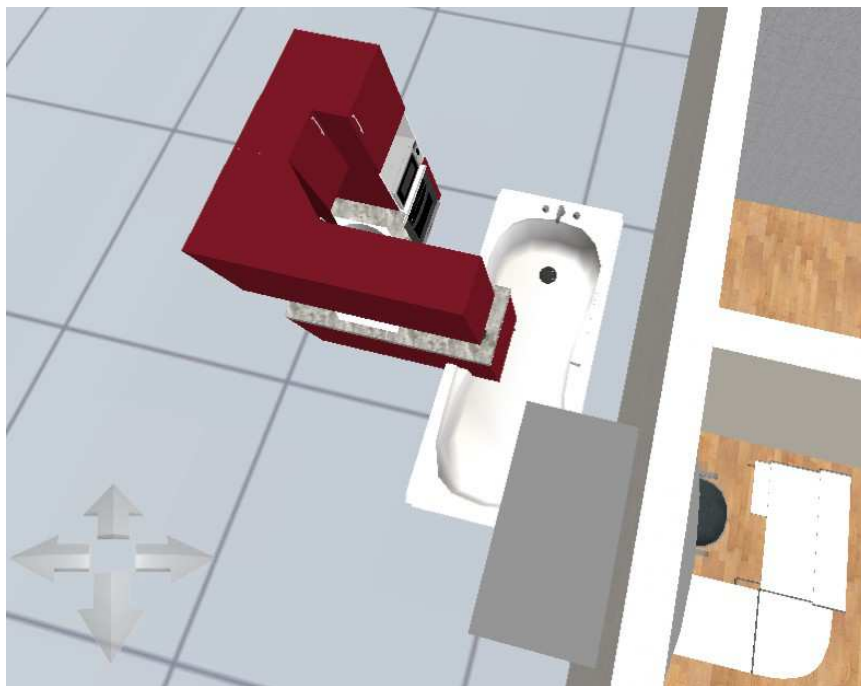


Figure 2-8

2.3.1.3 Item library

The application contains hundreds of generic furniture models. Some are available only in the paid version. They are divided into 4 categories, based on the room where they are usually placed. Each category is displayed as a long scroll list with object previews and brief descriptions. The objects are added to the scene by dragging from the item library (Figure 2-9).



Figure 2-9

2.3.1.4 Disadvantages/Shortcomings

- No editing and object manipulation in the 3D view, only changing textures of structural elements
- No grouping and selecting multiple objects
- No collision detection
- The program is unstable and crashes after certain device events

2.3.1.5 Advantages

- 2D room editing/drawing can be learned quickly
- Responsive and fast 2D mode, with the usual gesture mapping

2.3.2 Home 3D

2.3.2.1 Navigation in scene

The application comes with two 2D modes. One is used for creating the room layout (Figure 2-10). The second one is for decorating the rooms. Unlike in the commonly used iPad applications, the scene is moved with two fingers, but rooms are moved around with one. The pinch gesture zooms in the scene, and double tap changes the zoom to a predefined level.

Two 3D modes are also present in the application; the "Dollhouse" and the "Walkthrough" mode. In the "Dollhouse" mode, the user can add objects and edit the textures/materials. The camera view angle is rotated with one finger (pan gesture). Two

fingers are used to move the scene. The pinch gesture zooms in/out. The rotation gesture, with two fingers, is not used.

Furniture can be added in the 3D mode. When an object is selected, the meaning of the touch gestures changes and is applied to the selected object. The motion in the "Walkthrough" mode is partly controlled by a virtual joystick (Figure 2-11).

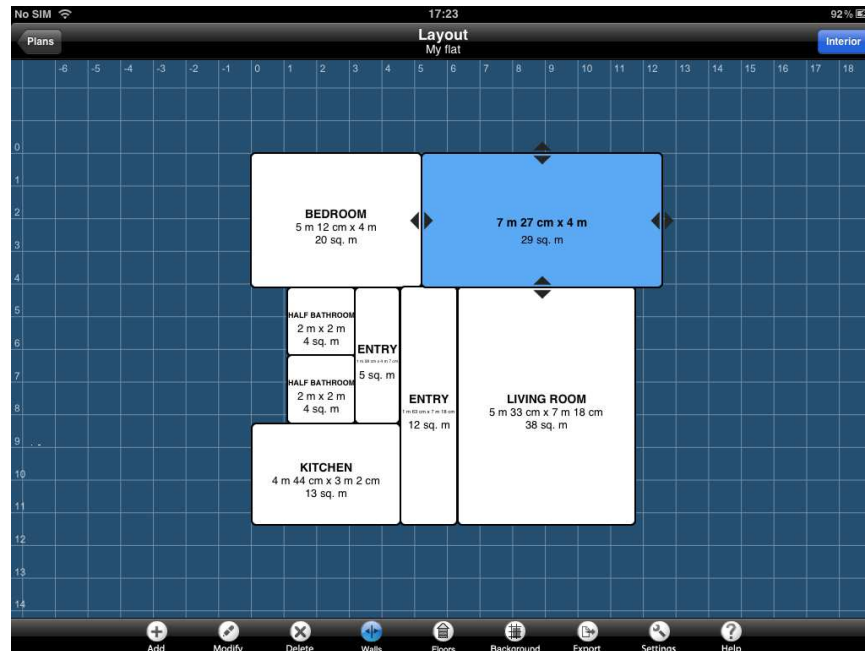


Figure 2-10



Figure 2-11

2.3.2.2 Manipulating items

Furniture models can be selected in both 3D modes. After selecting an object by tapping, it can be moved around in the same height level with one finger. However the manipulation is not direct. The item is not moved to the position under the finger, instead it is displaced by the same relative distance as the pan gesture, regardless of the current rotation of the 3D view. It depends on the displacement in the 2D coordinates of the tablet screen. This results in changing the expected direct manipulation behavior in the 3D mode. Direct manipulation is possible only when the camera displays the room from the top, thus reducing it to a 2D view. The height of the selected objects can be altered by a two-finger pan, without enforcing the presence of a support structure (Figure 2-12).



Figure 2-12

2.3.2.3 Item library

The item library includes various materials/textures and approximately 100 generic furniture items. The model library occupies the whole screen (the iOS Split View element). Item names and descriptions with preview images are laid out in a grid in the right part of the screen. Item categories are listed in the left part of the library (Figure 2-13).

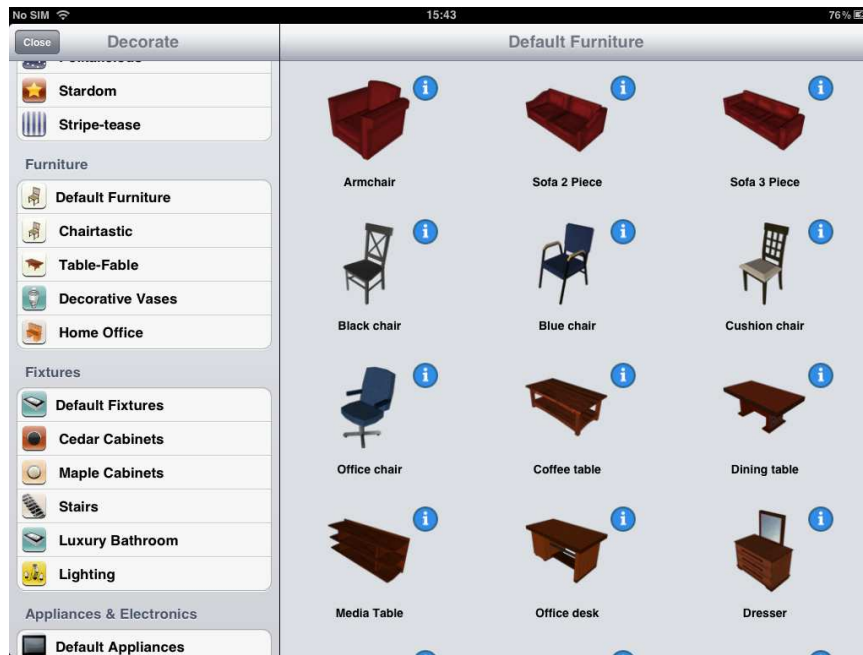


Figure 2-13

2.3.2.4 Disadvantages/Shortcomings

- Very counterintuitive gesture bindings (pan gesture with one finger has different meanings in each view mode)
- Gesture bindings are inconsistent; they change in different views and states
- Can not add non-rectangular rooms correctly (rooms are not merged after removing walls)
- Furniture is tied to the room where it was placed, can not be moved to other rooms
- Using relative displacement of the finger for moving objects in 3D mode, regardless of the current camera position; breaks direct manipulation

2.3.2.5 Advantages

- Extensive furniture model library
- "Superimpose" blueprints, import a blueprint and create a layout over it
- Direct file import/export from Dropbox
- Support for multiple floors

2.3.3 Living Room for iPad

2.3.3.1 Navigation in scene

The last tested application does not support any 3D mode. All editing is done in a 2D view (Figure 2-14), where structural elements, such as walls, doors and windows, and furniture can be added. To zoom in, one can use pinching, and one finger to move around by panning.

2.3.3.2 Manipulating items

Objects can be added and moved around the scene by dragging them with one finger. After tapping and object, nudge buttons in the left corner of the screen can be used to nudge it in

any direction. Resizing is available after selecting an object by dragging the resize handles around the object. Rotation is done by twisting the selected model with two fingers.

Double-tapping an object displays their properties in an info palette, with object, color and texture information. The info palette is crucial for object manipulation; their z-order (putting behind other objects), look, numerical size and rotation are edited there. Removing objects is also done in the info palette.

The user can move a single object in the scene or all the objects at once by selecting the "Move Everything" action. Arbitrary objects can not be selected at once to apply the same actions to the group.

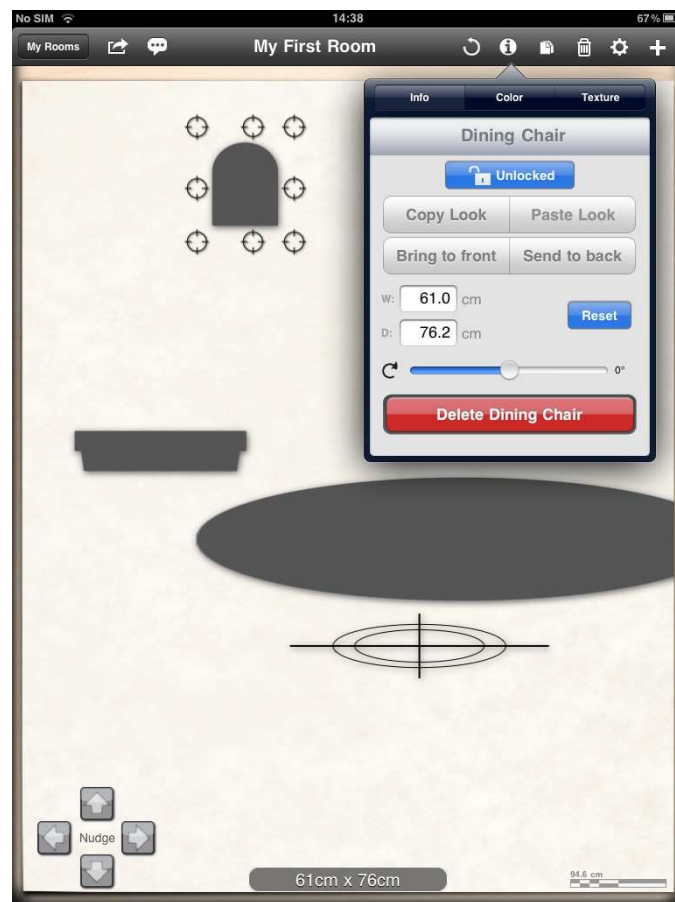


Figure 2-14

2.3.3.3 Item library

Objects are displayed in scrollable tray, which appears after pressing the “+” button, they are arranged alphabetically.

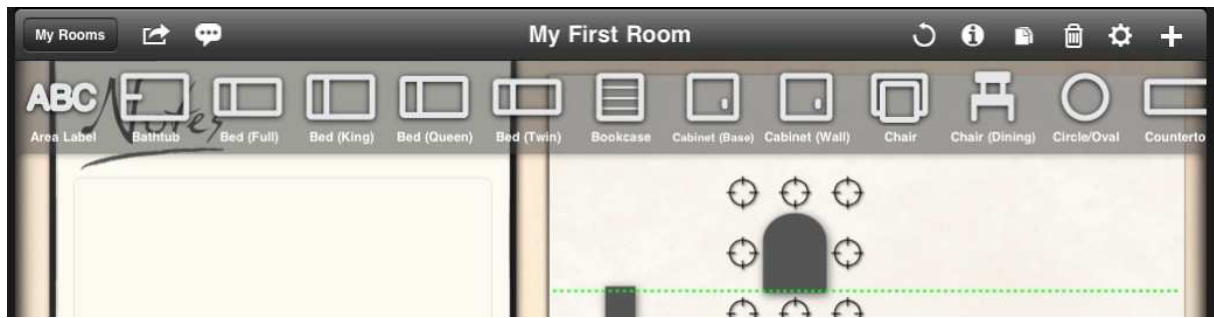


Figure 2-15

2.3.3.4 Other features

Custom textures from photos can be added to objects in the info palette. When holding the iPad horizontally in the room view, notes can be added to the notepad and stored along with each room. The output of the program is an image or PDF file.

2.3.3.5 Disadvantages/Shortcomings

- Objects can be placed over each other in the 2D view
- The object shapes do not resemble real-world furniture
- Models can be stretched to very unnatural shapes
- No 3D editing or 3D view modes
- The pinch gesture zooms the whole room, even when an object is selected, however the rotation gesture rotates the selected model, not the room
- No collision detection or way to join wall segments
- The markers of selected objects are generic, not icons depicting their actions (resizing in the desired direction)

2.3.3.6 Advantages

- Elegant main screen with an overview of previously edited projects
- Lines assisting the alignment of items

2.4 Conclusion

All the desktop apps share the same inherent issues with being hard to navigate in. The productivity of the user certainly gets better over time as one gets accustomed to the interface. Professionals, who use the products on a daily basis, achieve higher productivity by learning useful keyboard shortcuts, which can for example toggle between different modes, so the same motion of the mouse yields different results than rotation or panning. However a steep learning curve is not well suited for the occasional amateur user.

The iPad apps, which should be solving these problems with navigation in 3D still fail to capture the main advantage of the new platform. In the following chapters, ways of exploiting the capabilities of tablets to address these problems are discussed.

The observations that generally hold true for the tested iPad applications:

- The mapping of gestures to actions is inconsistent within the applications
- The commonly used gestures from 2D apps like maps, web-browser do not translate consistently to the 3D environment
- Direct manipulation is crucial for a natural user experience

There is certainly much room for improvement in many areas and functions like moving the furniture models around the scene or drawing of functional elements.

3 Platform Description

This chapter describes the context of the work; the iPad platform and the underlining technologies with a quick description of the Objective-C programming language, which should make the understanding of the implementation and design details of the thesis clearer.

3.1 The iPad

The iPad is a line of tablet computers developed by Apple Inc. The first one was introduced in 2010. The latest 3rd generation iPad comes with a multitouch 9.7 inch display with the resolution of 2048x1536 pixels. The CPU is an Apple A5X with the ARMv7 instruction set. The tablet comes with 1 GB RAM and a PowerVR 5GX554 GPU. The GPU supports OpenGL 2.0 ES. All models are equipped with a Wi-Fi network adapter.

The iPad runs the iOS mobile operating system, which also powers the iPhone and the iPod touch devices. The most recent version is iOS 6.1.

3.2 Objective-C and Cocoa Touch

Objective-C is the primary object oriented programming language for Apple platforms, the Mac OS X and the iOS. The language is a superset of C programming language. It extends the C language by providing syntax for defining classes, and methods, as well as other constructs for dynamic extension. The current version of the language is 2.0.

Nearly all concepts in Objective-C will be familiar to programmers with C, C++, Java, C# experience. Some more dynamic features will be familiar to Smalltalk, Lisp, or JavaScript users.

The public declaration of classes and functions reside in header files with “.h” extension, implementation files typically use the “.m” extension, or “.mm” extension if the file contains additional C++ code.

```
#import "ISWallManagerDelegate.h"
#include "PostProcess.h"

@interface ISArchView : Isgl3dBasic3DView <ISWallManagerDelegate> {

@private
    Isgl3dNode* _container;
    UIPinchGestureRecognizer* _pinchGestureRecognizer;
    UIRotationGestureRecognizer*
    _rotationGestureRecognizer;
}

@property (strong, nonatomic) ISShape* next;

- (void)drawWall:(UIBarButtonItem*)sender turnDrawingOn:(NSNumber*)drawingOn;
- (void)deleteWall:(UIBarButtonItem*)sender
    turnDeleteOn:(NSNumber*)deleteOn;
+ (ISVerticesInFaceEnumerator*)enumeratorForFace:(id)face;
```



```
@end
```

Code 3-1

A sample declaration of a class interface (the “.h” file) is shown in Code 3-1. The class `ISArchView` inherits from class `Isgl3dBasic3DView` and implements the `ISWallManagerDelegate` protocol (interface).

The `#import` directive is used for including Objective-C files, files containing C function declarations are included with the `#include` directive.

Unlike methods, instance variables can be private. The preferred way to declare variables is with the `@property` identifier, which generates the getter and setter method declarations for better encapsulation; however, for purposes of inheriting private or protected variables, the direct access to variables is still used. This part of the platform is confusing and there are further implications and benefits for both approaches.

Method names are usually very verbose, which is useful for documentation purposes. The main IDE, XCode, does a good job in assisting with code editing to save typing.

The first method in the example:

- Is private (the “-“ sign)
- Does not return any value (`void`)
- Takes two parameters, which are pointers to objects (`UIBarButtonItem*`) and (`NSNumber*`)
- The method is referred to in text as `drawWall:turnDrawingOn:`

Analogically to the “-“ sign, the “+” sign declares a class method, which can be invoked directly on the class. This causes ambiguities when drawing UML diagrams, where the “-“ and “+” denote the visibility of methods (public, private). As there are no private methods in Objective-C, the signs in UML diagrams in this thesis refer to instance and class methods.

The reason for not having separate categories for private and public methods is the fact that inter-object communication is described in terms of sending messages from one object to another. This is similar to calling methods of objects in other languages. The difference is in the dynamic dispatch and processing of messages. There is a runtime, which takes care of processing a message, instead of invoking methods known at compile time. This means, that any arbitrary message can be sent to an object, it does not need to respond to it. The objects can be queried to determine whether they respond to a certain message.

```
ISWallMesh* wallMesh = [[ISWallMesh alloc] initWithGeometry:wallLength
height:1.5f depth:0.1f nx:20 ny:20];

Isgl3dNode* node = [_container addChild:_currentWallNode];
node.rotationY = Isgl3dMathRadiansToDegrees(rotation);
```

Code 3-2

The Code 3-2 example shows how messages are sent to objects by enclosing the target object and the message in square brackets. C and C++ functions are invoked with the usual parentheses. If an object has a declared property, it can be accessed with the shorthand dot notation `node.rotationY` or with the square bracket syntax `[node rotationY]`.

Objects are allocated and initialized in two separate steps. They are always called together as `[[Class alloc] init]`. Unlike in C++ every object is allocated on the heap, the stack can not contain any objects.

In iOS 5.0 Automatic Reference Counting (ARC) was introduced, which simplifies memory management. In earlier versions the programmer had to perform the reference counting manually and make explicit calls to retain object count, and release/decrement the count when appropriate. Garbage collection of no longer referenced objects is not supported.

Other, more advanced features of the language and the operating system are introduced in later chapters in the context where they are employed.

- Lexical closures/Blocks – chapter 4.4.8
- ARC – chapter 4.1

Some design patterns that are not parts of the language are also discussed in later chapters. They are usually the best way to achieve certain behavior and describe how one should choose classes and methods. Objective-C and the Cocoa/Touch frameworks use only a few patterns, as the language itself offers good ways of abstraction. The patterns discussed are:

- Action-target – chapter 4.2.2
- Delegate – chapter 4.4.6
- Enumerator – chapter 4.4.3





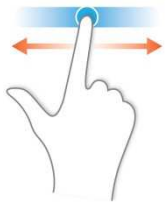
These design patterns are used throughout the application frameworks (libraries). In addition to the three listed patterns, the user interface classes use the Model-View-Controller pattern. The way it is understood and implemented by Apple, and how it differs from implementation of other vendors is discussed in detail in the online documentation [4]. Objective-C is not widely used outside of the Mac and iOS platforms. One of the reasons is the strong reliance on the Cocoa and Cocoa Touch application frameworks, which include collections, UI elements, etc.

3.3 Gestures

People make specific finger movements, called gestures, to operate the multitouch interface of iOS-based devices. For example, people tap a button to activate it, flick or drag to scroll a long list, or pinch open to zoom in on an image.

The multitouch interface gives users a sense of direct manipulation of onscreen objects. People are comfortable with the standard gestures because the built-in applications use

them consistently. Their experience using the built-in apps gives people a set of gestures that they expect to be able to use successfully in most other apps. All the gestures are expected to work the same; regardless of application.

Gesture	Action	
Tap	To press or select a control or item (analogous to a single mouse click).	
Double tap	To zoom in and center a block of content or an image. To zoom out (if already zoomed in).	
Touch and hold (Long press)	In editable or selectable text, to display a magnified view for cursor positioning.	
Drag (Pan)	To scroll or pan (that is, move side to side). To drag an element.	
Flick	To scroll or pan quickly.	

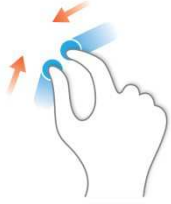

Pinch	Pinch open to zoom in. Pinch close to zoom out.	
Rotating	(fingers moving in opposite directions)	
Swipe	With one finger, to reveal the Delete button in a table-view row or to reveal Notification Center (from the top edge of the screen). With four fingers, to switch between apps on iPad.	

Table 3-1

Table 3-1 contains the gestures available on iOS devices. The table is an extended version of the overview in iOS Human Interface Guidelines [5].

The first three gestures are discrete; they only send one message to the target object on completion. Continuous gestures involving the movement of fingers send action messages to the target object in short intervals until the multitouch sequence ends [6].

Some gesture recognizers are prebuilt in the iOS. They do not return merely the position of the fingers, but also do the computation necessary to detect the rotation angle for a rotation gesture or a zoom level change for a pinch gesture. For more complex cases custom recognizers can be built by processing raw finger positions.

3.4 Conclusion

The iOS operating system, Cocoa Touch frameworks and the Objective-C have many powerful features, many nice features, and some confusing ones. The advantage is that the high consistency with the libraries and frameworks. This is a result of the language evolving simultaneously with large projects, the NeXTSTEP, Mac OS X, and the iOS.

4 Design and implementation

This chapter discusses the way certain features are implemented in the application. It involves both the software design and implementation steps. Separating them would

suggest a waterfall implementation model, which would not reflect the real way of development especially in the case of constantly trying out different approaches.

I decided to discuss the techniques I used during the development of the application, with a description of how they relate to the functions from the user experience standpoint. The discussion of the implementation also includes software development ideas for the iOS platform that I consider useful for the future development of the application.

4.1 Scene graph – iSGL3D

The iPad supports OpenGL ES 2.0 for rendering 3D graphics. My first attempt to build the application was by using the GLKit framework. The framework contains a pair of view and view controller classes; these can be incorporated in an application through the interface designer of XCode. The classes setup the OpenGL context and rendering canvas. With this setup, it is fairly simple to render geometric primitives. However one must be aware, that all the data has to be stored in VBOs (vertex buffer objects) and rendered with shaders.

The next step was incorporating models in the application (Figure 4-1). I used the Assimp library [7], which can import 3D models of several different formats. The library is written in C++ and also has a C API. Even though it took a long time to figure out how to compile and feed input files to the Assimp library, and get the mesh data loading correctly, I later had to abandon this solution in favor of a better approach with the PowerVR library (see chapter 4.5).

To be able to move the camera around and position the models, I built a few helper classes, which started to resemble a very rough scene library. In order to save time, I examined open source scene graph libraries for iOS, which could be incorporated in the project.



Figure 4-1

The one with the least shortcomings was iSGL3D [8]. However, it was far from ideal.

The framework is written for iOS devices in Objective-C and comes with a permissive MIT license. The framework is relatively small in size, which obviously limits the functionality provided. On the other hand, it is an advantage for understanding and extending it. One of the shortcomings of iSGL3D is the manual reference counting memory management technique it uses. This approach is deprecated in iOS5. Although XCode comes with an automatic utility to transform manual reference counting to ARC (automatic reference counting), it fails for certain cases. A deeper study of some memory management concepts was needed.

The other problem was an older project structure with no support for the user interface builder in XCode. The extension with storyboards is discussed in chapter 4.2.1.

The public API of the iSGL3D is reasonably well documented and simple scenes can be populated with geometric primitives, lights and textures with a few lines of code. The internals of the system come with little comments, it is therefore necessary to read through the source code of the library when extending it.

4.2 Menu

In order to make the application feel more like a built-in iOS application, I added a top menu (Figure 4-2) to it. The user interface guidelines suggest using as little buttons and menus in an application as possible [9]. The preferred way is to make objects on screen interactive.



Figure 4-2

There are functions for which the top menu is the best place to reside:

- Adding furniture models
- Saving and loading scenes
- Wall editing tools

4.2.1 Storyboards

The menus are stored in user interface files. They can be easily edited with the XCode IDE.

When an application is created from scratch in iOS 5, it supports Storyboards, which are a convenient way to represent the user interface and transactions between different screens in an application. They also reduce the glue code need to transition between different screens in the application.

The iSGL3D project was written for an older version of the operating system, where the application stored a nib user interface file for each view controller. On the other hand a storyboard behaves like a collection of these files. A single storyboard can hold the

interface information for multiple view controllers. Another advantage is the flexibility for creating new views with XCode.

The iSGL3D project does not support storyboards as it was created to be compatible with the fourth version of iOS. Applications built with iSGL3D and similar frameworks are usually games and contain no views other than the one for OpenGL. To make the application feel more consistent with the overall iOS experience, I extend the project to support storyboards and used generic elements of iOS interface as a Navigation Bar, Table Views and other prebuilt UI controls.

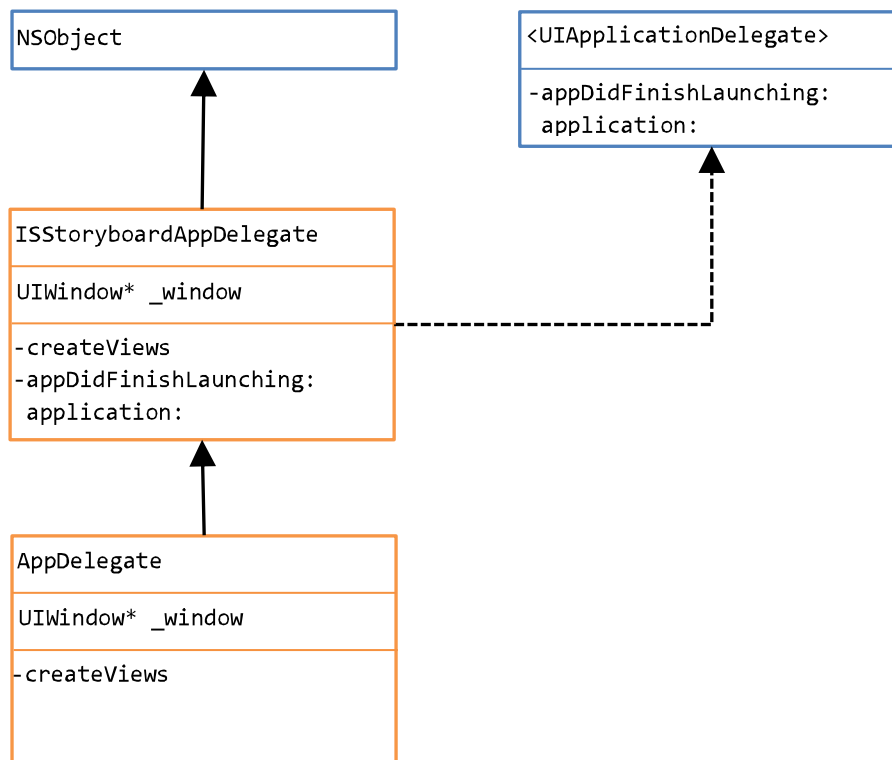


Figure 4-3

The new structure of the classes is shown in Figure 4-3. This is reflected in changing the main entry point of the application by passing it the name of the new `AppDelegate` class (Code 4-1).

```

// ISMain.m
#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {
    @autoreleasepool {
        UIApplicationMain(argc, argv, nil, @"AppDelegate");
    }
    return 0;
}
  
```

Code 4-1

In the older versions of the OS a `MainWindow.xib` file existed. This nib contained the top-level `UIWindow` object, a reference to the App Delegate, and one or more view controllers. With the storyboard version `MainWindow.xib` is no longer used and an App Delegate had to be created.

Previously, the application delegate was responsible for initializing the user interface – the main view and view controller. Now, the storyboard instantiates the first view controller from that storyboard and puts its view into a new `UIWindow` object. This view is then controlled by the first view controller, in this case the `ISViewController`. All these information are stored in the `Info.plist` file and can be edited in the user interface designer.

The main view – which became the main window of the application, contains the menu, its subviews (the buttons) and the OpenGL canvas. The `ISViewController` is now connected to its views (via outlets/pointers). It can communicate with the menu buttons and receive callbacks. This mechanism is known as Target-Action communication.

When this part of the setup is over, the `AppDelegate` is informed by invoking the `applicationDidFinishLaunching` method.

```
// ISStoryboardAppDelegate.m
Isgl3dEAGLView* glView = (Isgl3dEAGLView*)[[self.window rootViewController]
view];

// Set view in director
[Isgl3dDirector sharedInstance].openGLView = glView;
```

Code 4-2

The initialized view of the main window is then passed to the `Isgl3dDirector` in order to do the necessary setup of the EAGL OpenGL context and use it later for rendering.

4.2.2 Target-action mechanism

The communication between the menu buttons and `ISViewController` adheres to the target-action mechanism used throughout the Cocoa Touch frameworks. I implemented the same communication pattern in several places in the application.

Cocoa Touch uses the target-action mechanism for communication between a control and another object [10]. This mechanism allows the control to encapsulate the information necessary to send an application-specific instruction to the appropriate object. The receiving object – typically an instance of a custom class – is called the target. The action is the message that the control sends to the target. The object that is interested in the user event (the target) names the action.

Events by themselves are not enough to identify the user's intent; they merely tell that the user tapped a button. The target-action mechanism provides the translation between an event and an instruction.

A target is a receiver of an action message. A control holds the target of its action message as an outlet. The target is an instance of a class, which implements the appropriate action method.

One of the places where I implemented the target-action mechanism is the the ISArchView and ISViewController connection, to inform the ISArchView, that the current tool has changed (e.g. “wall drawing mode” to “wall delete mode”).

The AppDelegate class instantiates ISArchView and has access to the views in the main window, therefore it is the place where the target and the action, that should be called after a drawing tool/mode is selected. The target is the pointer to the instance of ISArchView and the action is a selector, which can be roughly thought of as a function pointer in C.

```
// AppDelegate class
@implementation AppDelegate

- (void)createViews {
    // Create view and add to Isgl3dDirector
    Isgl3dView *view = [ISArchView view];
    view.displayFPS = YES;
    [[Isgl3dDirector sharedInstance] addView:view];

    UIViewController* viewController = [self.window rootViewController];

    If ([[self.window rootViewController]
respondToSelector:@selector(setToolbarButtonsTarget:)]) {
        [viewController performSelector:@selector(setToolbarButtonsTarget:)
withObject:view];
    }
}
```

Code 4-3

4.3 Gestures

Besides the few buttons in the main menu of the application, gestures are the primary way of interacting with the applications. Designing them is a non-obvious task, as there are many degrees of freedom to the way fingers can move on the screen as compared to a discrete event of clicking a button, moving a slider, or selecting a menu item. A control object for these discrete events would recognize a single physical event as the trigger for the action it sends to its target. In iOS, there can be more than one finger touching an object on the screen at one time, and these touches can even be going in different directions.

4.3.1 Designing the gestures

The most important goals in mind for design the gestures were:

1. Use them in the same way as built-in applications do
2. Use them in the same way as other widely used applications do
3. Make them additive

The goals often implied contradictory approaches, which can be seen in the analysis part of the thesis. The first two goals describe the need to find widely used metaphors. If maps are used by 10000x more people than another application with a different gesture binding, it is

more likely that the users are more used to it, even if the other approach was better for a user without previous experience.

The aim for additive gestures means, that one gesture naturally extends to another. As the analysis uncovered, the current applications often break this approach. A counter example to an additive gesture would be using one finger to rotate the scene and two fingers to pan. An additive version would be using one finger to pan and two fingers to zoom in, as the gesture that begins with pan extends into a pinch.

4.3.2 `UIGestureRecognizer`s

When a user puts his finger on the screen, he touches more than a single point. The area is usually an oval. The position and the angle of the oval are used to estimate where the user meant to touch. The operating system does this part of the processing and returns a single point with x and y coordinates, however that is not the only point that the user has touched. This has to be accounted for when the exact position is needed. In cases like drawing walls and connecting them, the touch distance can be tens of pixels away from the desired value. The point the OS returns is called a touch.

`UIGestureRecognizer`s are a set of prebuilt objects that can be attach to `UIView`s. They receive the raw touch events that “bubbled” through the view hierarchy. Multiple gesture recognizers can watch touches on the same view. The view can still receive the raw touch events and do custom processing of them.

A window delivers touch events to a gesture recognizer before it delivers them to the hit-tested view attached to the gesture recognizer. Generally, if a gesture recognizer analyzes the stream of touches in a multi-touch sequence and does not recognize its gesture, the view receives the full complement of touches.

When a gesture recognizes, it fires one or more action messages to its targets (the Target-action mechanism discussed in chapter 4.2.2). These actions are fired in a nondeterministic order. When a touch begins, each gesture recognizer is given a chance to see it, again, in a nondeterministic order.

Each recognizer goes through a set of states (Figure 4-4). It begins in the state “Possible” and through analyzing the touches it transitions to another state. Throughout this transition actions are continuously fired for states “Recognized”, “Began”, “Changed”, “Ended”, and “Canceled”. Discrete gestures as tap or double tap fire only once. The continuous ones fire multiple actions (Figure 4-5) [6].

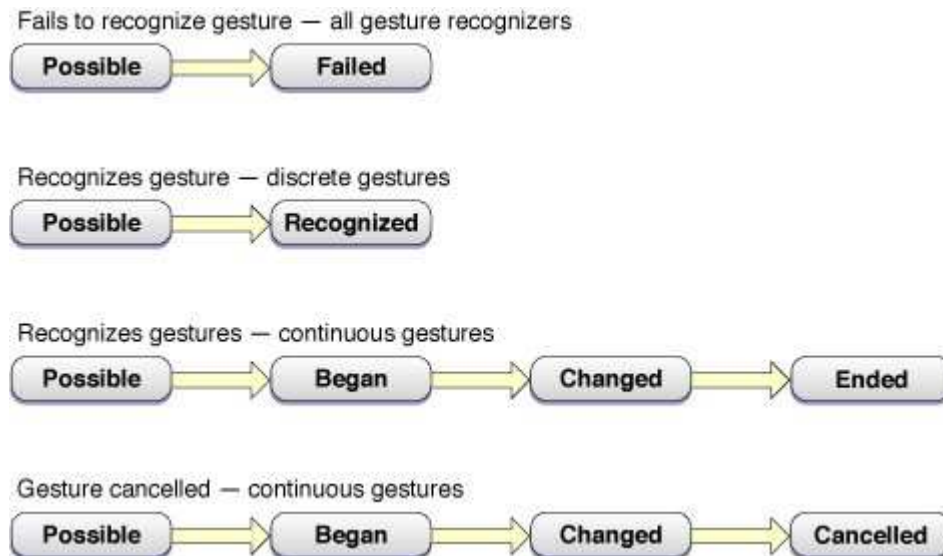


Figure 4-4

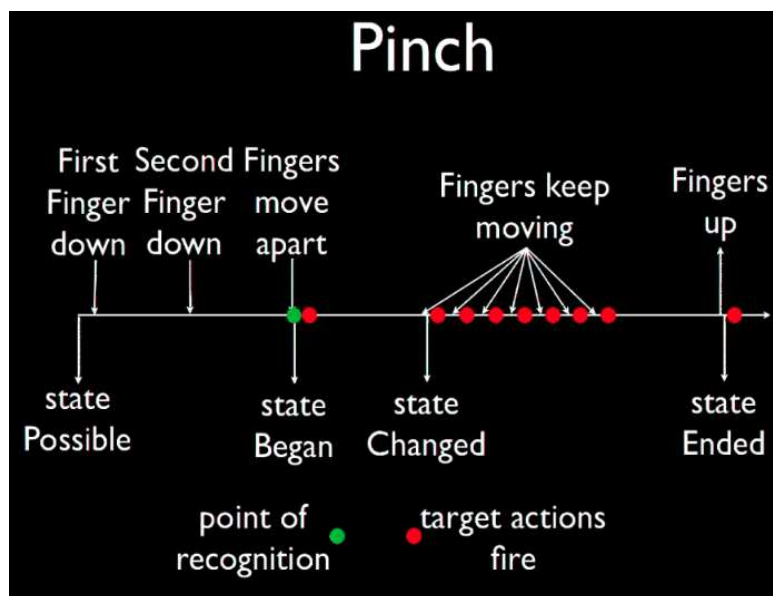


Figure 4-5

4.3.3 Touch events in iSGL3D

Touch events seamlessly propagate through the rectangular view hierarchy of the UIViews. The operating system does not have any means to do the same for a 3D OpenGL scene. The view element, which contains the scene, receives the touch events. The controller of the view has to decide on how to process these events.

The public APIs of the iSGL3D library are well documented; however there is little information about the internals of the system. In order to extend the library, I had to “reverse engineer” the event handling code, which uses fairly advanced design techniques like proxy objects.

```
Isgl3dCube * cube = [Isgl3dCube meshWithGeometry:1 height:2.0 depth:0.1 nx:1.0
```

```

ny:1.0];
Isgl3dMeshNode* meshNode = [_container createNodeWithMesh:cube
andMaterial:[Isgl3dColorMaterial materialWithHexColors:@"0x000000"
diffuse:@"0xf0f000" specular:@"0xffffffff" shininess:3.0f]];

meshNode.interactive = YES;
[meshNode addEvent3DListener:self method:@selector(furnitureTouched:)
forEventType:TOUCH_EVENT];
UITapGestureRecognizer* furnitureTap = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(furnitureTapped:)];

```

Code 4-4

The example Code 4-4 shows, how a node for storing a cube is set to interactive mode and how a touch and a tap listener are added to it.

The tap has no notion of the object it has hit. It only knows its internal state and the 2D position in the view. To overcome this limitation the touch event is registered for the object. A tap is a press and release sequence, a touch is a position of a finger that the UIView receives (and forwards to a gesture recognizer if any is registered).

The touched object is determined by rendering all the objects with solid colors in another rendering pass by calling the `renderForEventCapture` method. The `Isgl3dObject3DGrabber` assigns colors to the objects, subsequently the engine gets the touched object by taking the touch point coordinates, translating them into view coordinates and looking up the color information of the pixel with the `glReadPixels()` function. The Code 4-5 example shows a mutable dictionary container with the “color” values of the rendered scene graph nodes.

```

(lldb) po _activeObjects
(NSMutableDictionary *) $51 = 0x1e548600 {
    0x000001 = "<Isgl3dMeshNode: 0x1e549b20>";
    0x000002 = "<Isgl3dMeshNode: 0x1e549de0>";
    0x000003 = "<Isgl3dMeshNode: 0x1e549f70>";
    0x000004 = "<Isgl3dMeshNode: 0x1e54a100>";
    0x000005 = "<Isgl3dMeshNode: 0x1e54a2a0>";
    0x000006 = "<Isgl3dMeshNode: 0x1e54a430>";
    0x000007 = "<Isgl3dMeshNode: 0x1e54a5c0>";
    0x000008 = "<Isgl3dMeshNode: 0x1e54a750>";
    0x000009 = "<Isgl3dMeshNode: 0x1e54a900>";
    0x00000A = "<Isgl3dMeshNode: 0x1e54aa90>";
    0x00000B = "<Isgl3dMeshNode: 0x1e54ac20>";
    0x00000C = "<Isgl3dMeshNode: 0x1e54adb0>";
}

```

Code 4-5

To work with this event handling method, the classes listening for events have to store the results of a touch event and use them for further decision making. An example of two action listener methods for the `ISWallManager` class is in Code 4-12.

4.3.4 Pan to move

Panning is the basic transformation of the scene view. The camera’s position is translated based on the translation in the 2D view coordinates, which are “unproject” and intersected

with the drawing *xz* plane (see chapter 4.6.). The displacement between the last camera and “lookAt” positions are added to get the new one.

The advantage of this approach is the direct manipulation, which is the preferred way by the Human Interface Guidelines [12]. The same point stays under the finger during the whole motion.

Code 4-6 is an example of a pan translation in a `UIView`, which is a special case of a continuous gesture. Translation in the view is the relative displacement of the point initially touched in `StateBegan`, whereas the location in view is the absolute position. Translation can happen also in `StateBegan`.

```
2012-04-18 23:18:51.741 Architectura[7087:707] UIGestureRecognizerStateChanged
2012-04-18 23:18:51.744 Architectura[7087:707] translation in view: 47.000000,
2.000000
2012-04-18 23:18:51.747 Architectura[7087:707] location in view: 589.000000,
454.000000
2012-04-18 23:18:51.757 Architectura[7087:707] UIGestureRecognizerStateChanged
2012-04-18 23:18:51.758 Architectura[7087:707] translation in view: 45.000000,
0.000000
2012-04-18 23:18:51.760 Architectura[7087:707] location in view: 587.000000,
452.000000
2012-04-18 23:18:51.773 Architectura[7087:707] UIGestureRecognizerStateChanged
2012-04-18 23:18:51.775 Architectura[7087:707] translation in view: 43.000000,
-1.000000
2012-04-18 23:18:51.779 Architectura[7087:707] location in view: 585.000000,
451.000000
2012-04-18 23:18:51.789 Architectura[7087:707] UIGestureRecognizerStateChanged
2012-04-18 23:18:51.792 Architectura[7087:707] translation in view: 40.000000,
-2.000000
2012-04-18 23:18:51.795 Architectura[7087:707] location in view: 582.000000,
450.000000
2012-04-18 23:18:51.805 Architectura[7087:707] UIGestureRecognizerStateChanged
2012-04-18 23:18:51.808 Architectura[7087:707] translation in view: 38.000000,
-5.000000
2012-04-18 23:18:51.812 Architectura[7087:707] location in view: 580.000000,
447.000000
```

Code 4-6

4.3.5 Pinch to zoom

The pinch gesture recognizer returns the scale factor (Code 4-7), based on the distance between two fingers moving to or from each other. The scale is relative to the initial finger distance when the pinch recognizer recognized, therefore a scale > 1 can as well mean zoom out from the current camera view.

```
2012-04-19 13:11:16.172 Architectura[7909:707] pinch scale 2.194589
2012-04-19 13:11:16.498 Architectura[7909:707] pinch scale 1.474392
2012-04-19 13:11:16.541 Architectura[7909:707] pinch scale 1.329444
2012-04-19 13:11:17.740 Architectura[7909:707] pinch scale 0.988198
2012-04-19 13:11:17.756 Architectura[7909:707] pinch scale 0.984519
```

Code 4-7

The first working idea was to make zoom compatible with the direct manipulation paradigm – the points under fingers stay at the same place; this mostly true for 2D interfaces like the in-built maps application.

This approach is depicted in Figure 4-6. Both pinch moves account for the same displacement. Based on the perspective projection, the A movement would result in a smaller zoom than movement B.

In the implementation, the positions of both fingers in the scene were computed and the scene was transform accordingly. The points under the fingers were the same during the whole zoom phase. This would, in theory, make this approach ideal.

In real usage, the different pace of zooming felt confusing. Users tend to chain the zoom gestures rapidly one after each based as needed, based on the feedback from the device, similarly as pressing the arrow keys on a keyboard until the cursor gets to its position.

The better solution was to take the relative zoom level, regardless of the position of the touches, adjust its pace with a linear function as it seemed to fast, and along the direction from the camera center to the “look at” point/the center of the view.

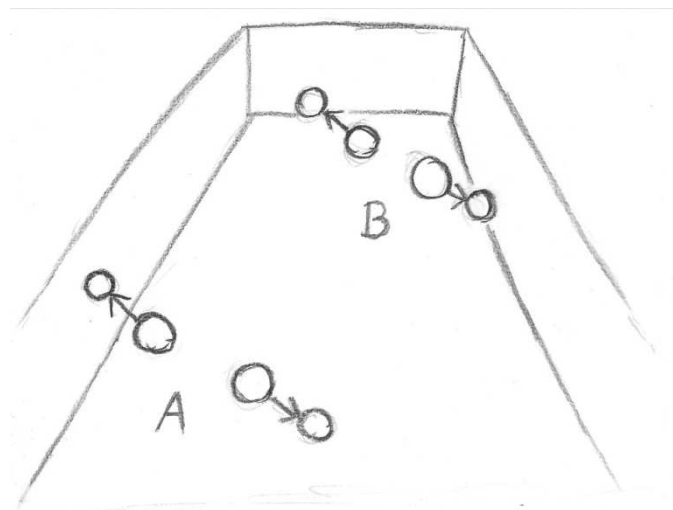


Figure 4-6

4.3.6 Rotation

The rotation functionality is implemented in a similar fashion to the zoom function. The difference is in the getting the angle size from the recognizer instead of the scale level. The angle is computed from the initial position of the fingers in the `StateBegan` mode.

The rotation is limited to moving the camera position in a plane parallel to the main *xz* drawing plane. This is necessary in order to preserve the semantics of the rotation gesture recognizer.

To make the experience richer, I tried to implement a trackball rotation mechanism around the center for the two finger pan gesture. Including this recognizer badly interfered with the other gestures (see chapter 4.3.7).

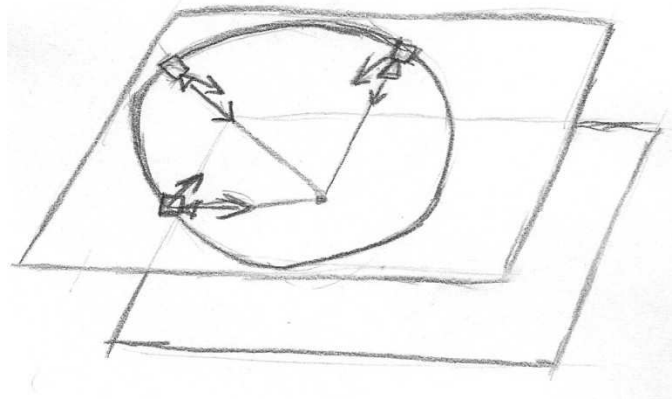


Figure 4-7

The camera positions in all the view transformations are based on the “look at camera” class `ISgl3dLookAtCamera`, which computes the view matrix based on the position of the camera, the point the camera is oriented towards, and an up vector. To make updating the eye and up position during animations and rotations easier, I created a subclass `ISLookAtAnimationCamera`, which computes the precise “up vectors” to prevent jiggles while interpolating the vectors during animation. The default way of computing the view matrix with the look at function does not require the “up vector” to be perpendicular to the view vector, as it only needs the plane they both lie in and a vector perpendicular to that plane.

4.3.7 Combining gestures

As mentioned earlier, one view can have multiple gesture recognizers attached, all of them can recognize simultaneously. This is seldom the desired behavior. Imagine a pinch gesture, with two fingers moving. The pan gesture would also recognize at this motion.

The default OS behavior is that no two gestures can recognize at the same time. This is also not desired, as one would likely combine a zoom and a rotation in one motion.

In order to successfully work with more recognizers the restrictions on them have to be rather strict as the number of possible combinations of touches is high. A new touch can appear at any time by placing another finger on the screen.

The two ways to restrict this behavior is by using:

- `<UIGestureRecognizer>`
- Define failure requirements

```
2012-04-21 17:32:00.289 Architectura[10395:707] gestureRecognizerA
UIPanGestureRecognizer gestureRecognizerB UIPinchGestureRecognizer
2012-04-21 17:32:00.292 Architectura[10395:707] gestureRecognizerA
UIRotationGestureRecognizer gestureRecognizerB UIPinchGestureRecognizer

2012-04-21 17:38:50.990 Architectura[10395:707] gestureRecognizerA
UIPanGestureRecognizer gestureRecognizerB UIRotationGestureRecognizer
2012-04-21 17:38:50.992 Architectura[10395:707] gestureRecognizerA
UIPinchGestureRecognizer gestureRecognizerB UIRotationGestureRecognizer
```

Code 4-8

The delegate (see chapter 4.4.6) receives pairs of recognizers and has to determine, whether they can recognize at the same time. The gestures have to be filtered carefully, not only based on their class, but also on their state, some of them might be in `StatePossible`, thus not recognized yet. An example of this is shown in Code 4-9.

The failure requirement is used to separate two gestures, which begin with the same motion. One of them is the pinch-pan dependency. The user does not put down both fingers at the exact same time to begin zooming. He first starts to move one finger, which could result in both a pan and a pinch. To make sure these two are not processed at the same time the application requires the pinch motion to fail before the pan begins. It fails, if a second finger does not come down shortly after the first one.

An analogue, not used in the application, would be the need for a double tap to fail before a tap recognizes.

```
- (BOOL) gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer
*)otherGestureRecognizer
{
    if ([otherGestureRecognizer class] == [UIPanGestureRecognizer class]) {
        if ([otherGestureRecognizer state] == UIGestureRecognizerStateBegan) {
            return NO;
        } else if ([otherGestureRecognizer state] ==
UIGestureRecognizerStateChanged)
            return YES;
        }
    }
    if ([otherGestureRecognizer class] == [UIPinchGestureRecognizer class]) {
        if ([gestureRecognizer class] == [UIPanGestureRecognizer class]) {
            if ([gestureRecognizer state] == UIGestureRecognizerStateBegan) {
                return NO;
            }
        }
        ...
        ...
        ...
    }
}
```

Code 4-9

4.3.8 Conclusion

The final design of the gesture handling could possibly be improved by writing a custom gesture recognizer and trying to incorporate the two finger pan in it.

An interesting finding is that the direct manipulation approach does not necessarily lead to the best results. It is more important to find the right settings of the zoom steps, than to do a mathematically precise computation.

4.4 Walls

The drawing of walls is a crucial part of the application. I have tried several ways of approaching this problem before settling for the current solution. The goal was to be able to draw walls without the need for a separate editor. The ideal result would be a consistent

experience with the other functions of the application, namely editing the furniture and moving around the scene. The view would also be rendered online, without the need to wait for the scene to be processed to a 3D representation.

4.4.1 The user interface

One of the shortcomings of the touch user interface is that there isn't a notion of the "mouse over" function, i.e. there isn't any cursor to be placed over an element without triggering an action. The first approach was to draw the wall between discrete points of user taps, this had two shortcomings, it was hard to draw straight walls and there were no means to specify the length of the wall.

A second approach was the "touch-pan-release". The pan gesture is a continuous move from the first point of the wall to its end point. The issue with this approach is that there has to be a way to distinguish between the user's intention to move around the scene and the intention to draw a wall. This is solved by adding a toolbar button, which can toggle between two modes – the "scene exploration mode" and the "wall drawing mode". When the user wants to move the view to the side, he has to switch modes first. This solved the problem with being able to see the wall while drawing, but addressed the problem of lengths only partially. The "look at" perspective camera distorted the lengths and made it hard to draw parallel walls. To overcome this issue, the camera in the "wall drawing mode" transitions to a top view position (Figure 4-8), which makes the basis vectors x and z (the plane in which the wall vertices are placed) perpendicular.

To delete existing walls, the user can open a context menu by tapping a wall and selecting the "delete" action. In the menu, various other properties like height and thickness can be adjusted.

It is hard to tap on a wall in the top view; therefore the "delete mode" uses the same camera position as the "scene exploration mode" (Figure 4-9).

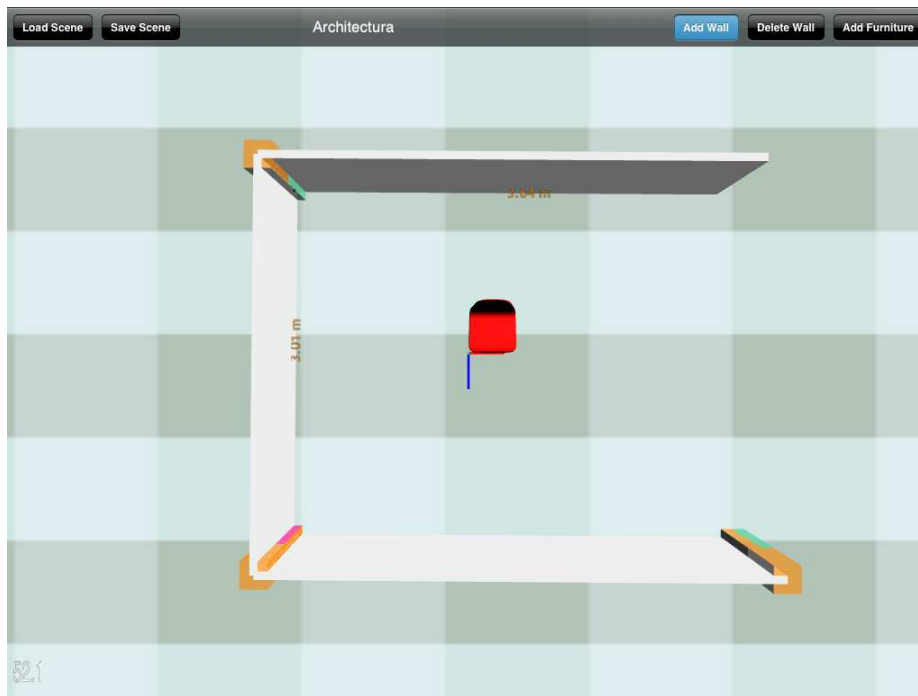


Figure 4-8

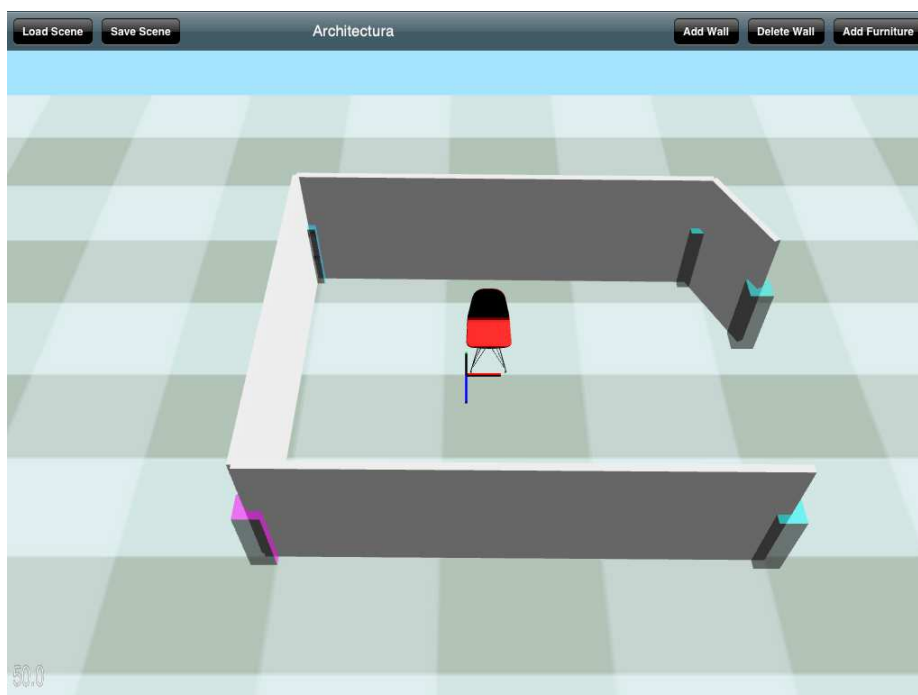


Figure 4-9

4.4.2 Winged-edge data structure

There are two sets of information to store for each wall. The first one is the geometry of the walls. This is passed to OpenGL with a precise position of the geometry vertices. The other set of information is the topology of the walls in the scene; more precisely, the topology of the walls, floors and rooms.

To extract the topological information the walls are represented as edges of a graph. The vertices are the places where the walls are joined and the floors are the faces next to the edges in the graph. By choosing this representation, a winged-edge data structure was formed (Figure 4-10).

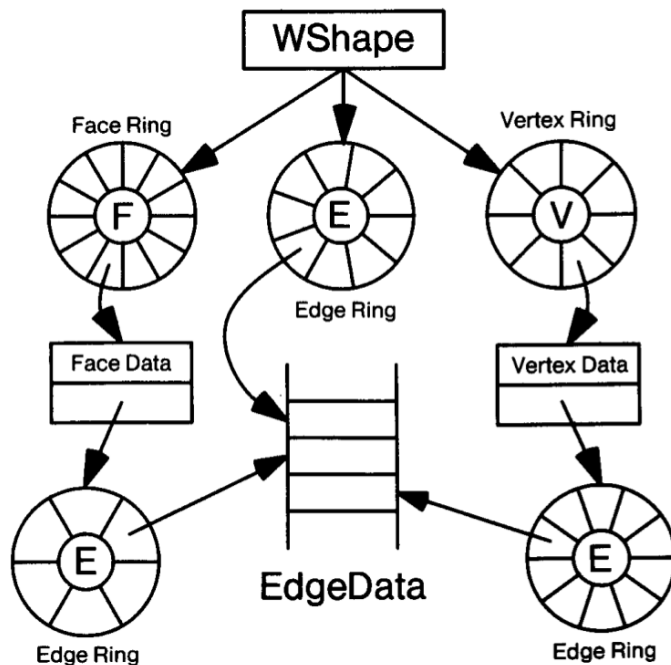
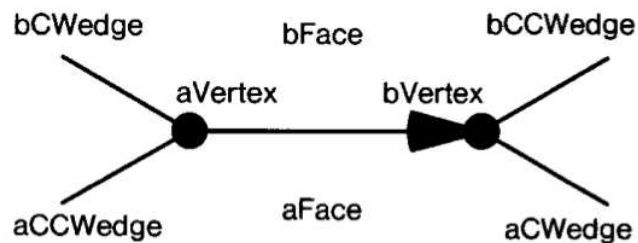


Figure 4-11

Figure 4-10

The winged edge data structure is based on the idea of an edge and its adjacent polygons. The architecture (Figure 4-11) of the winged-edge library is the one suggested by Pat Hanrahan and Andrew Glassner [13]. Faces, edges, and vertices each are stored in rings (doubly linked list). A WShape contains three rings, one each for faces, edges, and vertices. Each edge ring entry points to a data structure called WEdgeData, which contains the information for that edge. Each face contains an edge ring describing the edges around that face. Each vertex contains an edge ring of all edges around that vertex. All duplicate instantiations of an edge point to a single WEdgeData structure [13].

The Face Data points to a scene graph node representing the floor, the Edge Data has a pointer to a wall and the Vertex Data store the position of the vertex in world coordinates.

There is a lot of redundancy in the data structure, in order to make the queries easier. The toll is the need to carefully adjust the pointers in the structure.

Each ring is created as a doubly linked list of the class `ISEdge`, `ISVertex` or `ISShape`. By using the winged-edge representation, it is simple to obtain the topological structure of the walls. To build up the structure it has to be updated while drawing the walls (see chapter 4.4.7).

4.4.3 Enumerating the winged-edge structure

There are several queries one can send to a winged-edge structure. The most common are:

- List of edges for a face
- List of vertices for a face
- List of edges for a vertex

They are used in many places over the application. In order to have a common way of accessing them, I have created enumerators for the introduced cases.

The enumerators inherit from the `NSEnumerator` base class. The core method to implement is the `nextObject:` method. The enumerators like `ISRingEnumerator` or `ISRingDataEnumerator` can be efficiently requested by calling the `enumeratorForRing:` class method with the winged-edge element to be enumerated. Code 4-10 shows one possible usage scenario.

```
// ISFace class
- (NSEnumerator*)vertexPositionEnumerator {
    return [ISVerticesInFaceEnumerator enumeratorForFace:self];
}

// ISWallManager
NSEnumerator* verticesInFace = [face vertexPositionEnumerator];
Isgl3dVector3 room = Isgl3dVector3Make(0.0f,0.0f,0.0f);
int numberOfVerticesInFace = 0;
Isgl3dVector3 vertexPosition;
for (NSValue* vertexPositionObject in verticesInFace) {
    [vertexPositionObject getValue:&vertexPosition];
    room = Isgl3dVector3Add(room, vertexPosition);
    ++numberOfVerticesInFace;
}
...
...
```

Code 4-10

4.4.4 Geometry

The geometry of the walls and floors is stored in classes derived from the `iSGL3D` bases classes. The `ISWallNode` can be included in the scene graph rendering. In addition to storing the meshes of the wall and the textures, it adds properties for the positions of its two vertices. It can return the position of the vertices in its own coordinate system and more importantly in the world coordinate system. The world coordinates can be associated with the values in the winged-edge structure.

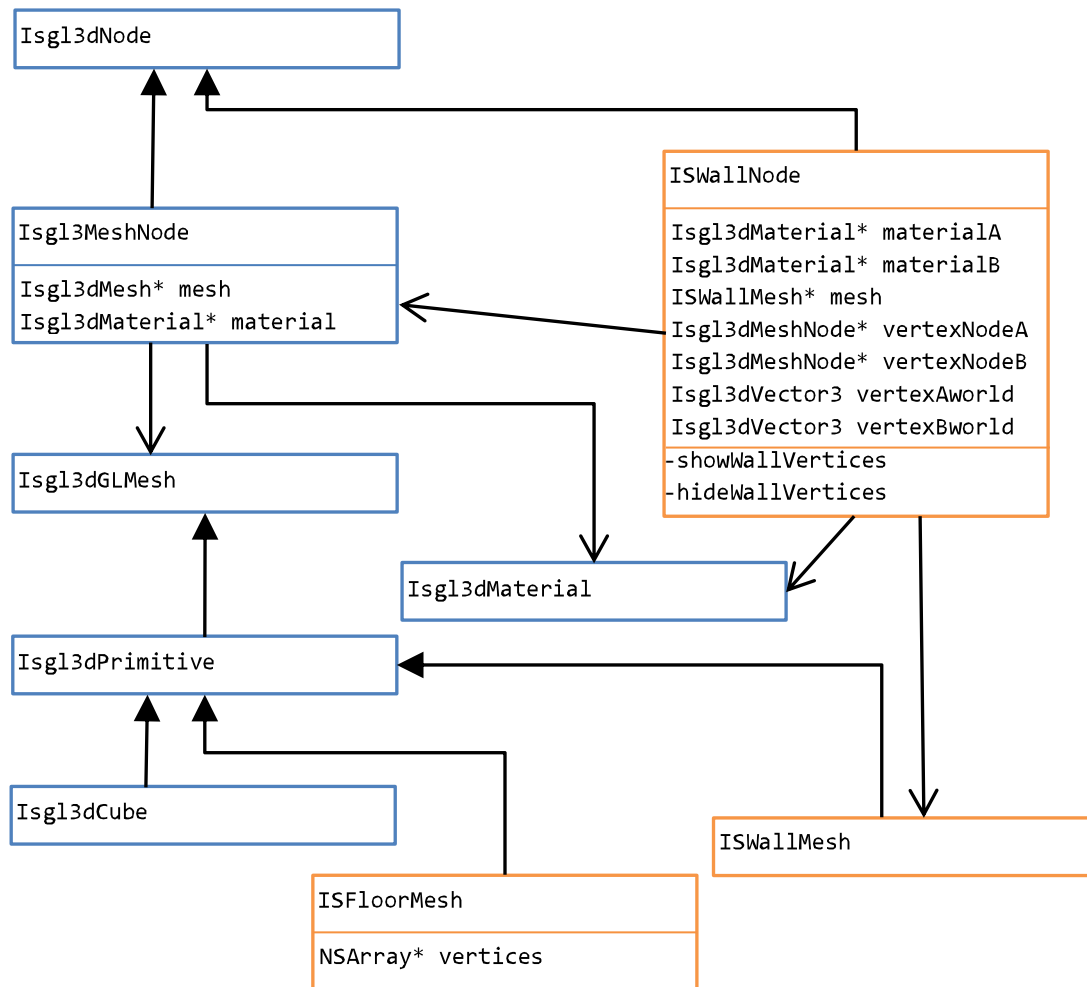


Figure 4-12

4.4.5 Drawing walls

The class responsible for drawing, deleting, and maintaining walls is called `ISWallManager`. Drawing a wall is not a discrete event; instead, it is a series of calls to the `ISWallManager` that have to appear in a certain order. This is captured by a state variable, which can be concisely depicted as a state transition system reacting to certain events sent from the `ISArchView` class (Figure 4-13).

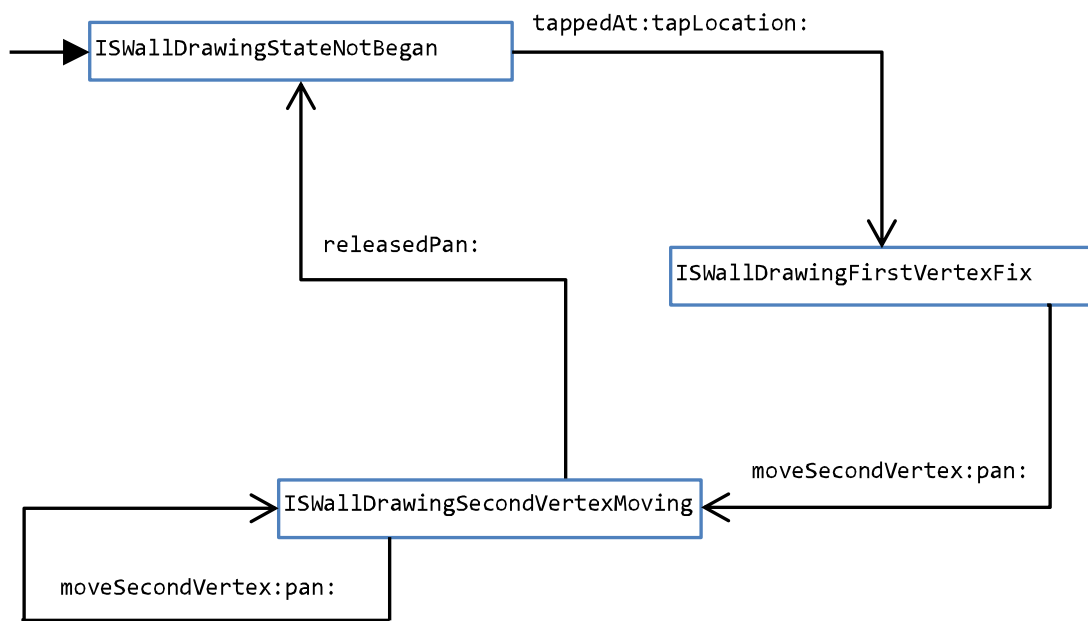


Figure 4-13

4.4.6 Removing walls and delegation

The result of tapping a wall is dependent on the currently selected tool. If the state is set to deleting walls a tapped wall is removed from the data structures. The `ISWallManager` is responsible for drawing and maintaining the state of the walls and processing events related to the walls. By default, it has no means of knowing the current state of the application and the events of the menu. This information is available to `ISViewController` and the `ISArchivView` classes.

The `ISArchivView` implements the `ISWallManagerDelegate` protocol and sets itself up as the delegate of the `ISWallManager` object (Code 4-11). The `ISWallManager` can seamlessly work without the delegate. If the delegate is present, it is asked whether the wall manager should now be removing walls (Code 4-12).

This type of delegation pattern is often used by the system frameworks. Examples can be found in various gesture recognizer and table classes.

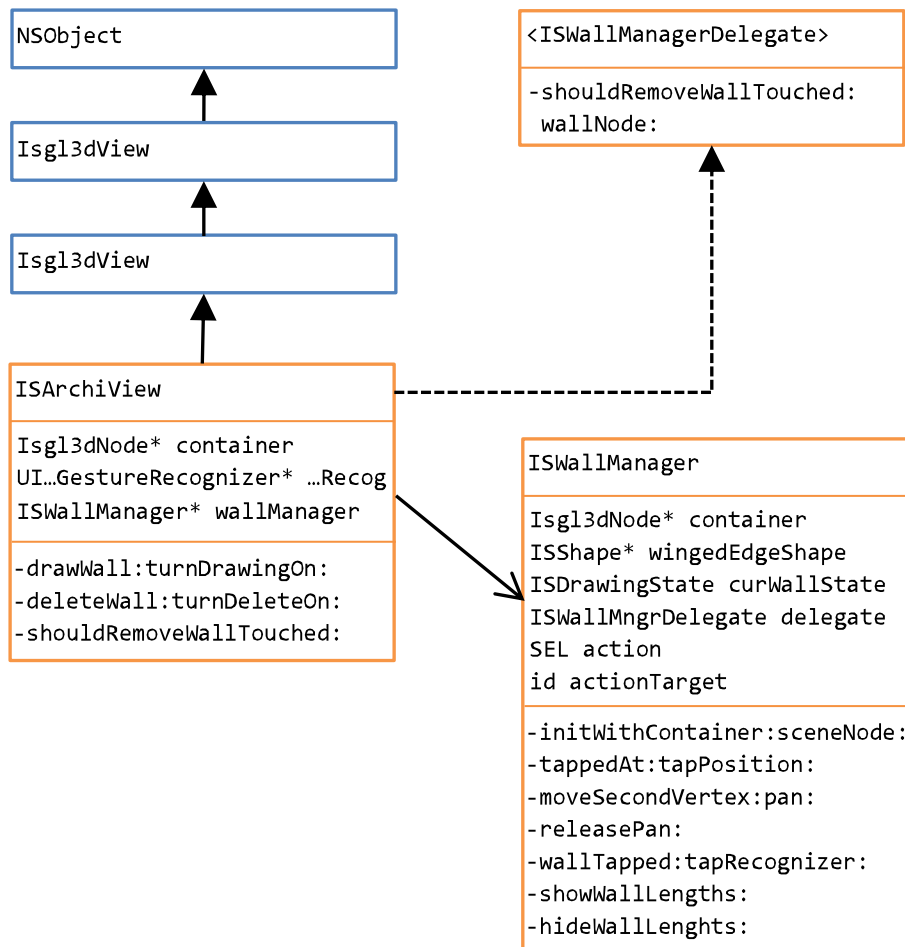


Figure 4-14

```

//ISArchiview.m
_wallManager = [[ISWallManager alloc] initWithContainer:_container];
_wallManager.delegate = self;

```

Code 4-11

```

//ISWallManager.m
//called 1st to indicate which wall was touched
- (void)wallTouched:(Isgl3dEvent3D *)event {
    _touchedWall = event.object;
    //if touched close to the vertex, draw wall
}

//called 2nd
- (void)wallTapped:(UIGestureRecognizer *)tapRecognizer {
    if (self.delegate) {
        if ([self.delegate shouldRemoveTouchedWall:_touchedWall]) {
            [_touchedWall removeFromParent];
            [self removeWallFromWE:_touchedWall];
            _touchedWall = nil;
        };
    }
}

```

Code 4-12

4.4.7 Creating rooms

The user is limited to drawing single walls. While drawing, the winged-edge data structure is constantly being updated with new edge, vertex and face information.

When a new vertex is drawn close enough to an existing vertex, the two edges are connected. This can lead to closing a circle and forming a room-like shape. In this case, a floor mesh can be added to the geometry.

The vertices and edges can form several disjoint components. As the number of the vertices is relatively low, it can be traversed to find a newly created room – a circle in the graph.

When a new room is formed, two situations can arise:

1. A face is formed, and there isn't any edge with an incident face (Figure 4-15)
2. A face is formed, and at least one of it's edge has an incident face (Figure 4-16)

In the first case, the new face is simply oriented counter-clockwise. In the second case the orientation of the shared edge is determined, and the new face is added in the opposite direction.

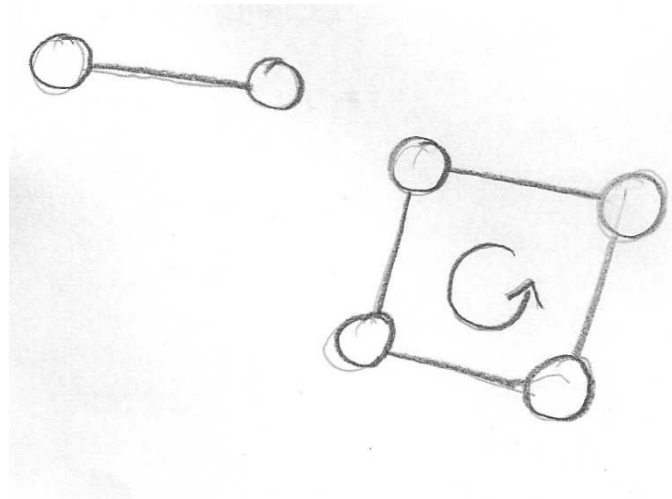


Figure 4-15

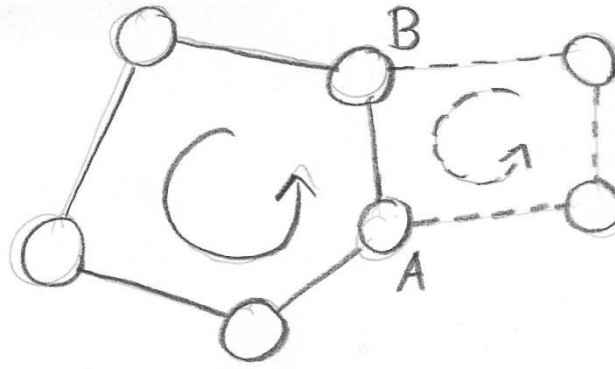


Figure 4-16

To traverse the edges, the BFS algorithm is used, because it finds the optimal solution, i.e. the shortest path between the vertices of the last edge that was added to the graph, leaving out the path with length 1.

To implement the BFS over existing data structures a technique called associative references [14][15] is used. Associative references, can extend any class at the runtime with an object instance variable. This is done without modifying the class declaration.

This is useful in order to keep the data structures unmodified for the other, parts of the program, or in when the source code is not available.

For the purpose of BFS a parent node is added to each vertex for extracting the path. The notion of a parent node and “traverse state” makes sense only in the context of BFS.

An example of the C language API to access associative references is shown in the context of the BFS search method (Figure 4-17).

```
- (NSArray*)checkNewFaceFormedBetweenVertex:(ISVertex*)vertexA
    andVertex:(ISVertex*)vertexB
    connectedWithEdge:(ISEdgeData*)edgeData {

    static char traverseState;
    static char parentVertex;

    ...

    NSMutableArray* queue = [[NSMutableArray alloc] init];

    objc_setAssociatedObject(vertexA, &traverseState, [NSNumber
numberWithInt:ISTraverseGoal], OBJC_ASSOCIATION_RETAIN_NONATOMIC);
    ...
    ...
    ISVertex* currentVertex = vertexB;
    objc_setAssociatedObject(currentVertex, &traverseState, [NSNumber
numberWithInt:ISTraverseClosed], OBJC_ASSOCIATION_RETAIN_NONATOMIC);
    ...
    ...
    currentVertex2 = objc_getAssociatedObject(currentVertex2, &parentVertex);
```

Figure 4-17

4.4.8 Blocks

To perform an action when a room is formed, the application exploits blocks instead of callback functions. Blocks [16] are functions or methods, which have access to their outer context, even when the outer context goes out of scope, which has several advantages to callback functions. In the Code 4-13 example the `^(ISFace* face)` the block is stored as a instance variable of the `_wingedEdgeShape`, which invokes the block, when a face is formed. Then the block can easily access the container of the scene through the `weakContainer` variable and add new elements (floor) to the scene graph.

```
// ISWallManager.m
- (id)initWithContainer:(Isgl3dNode *)sceneContainerNode {
    //TODO super initialization
    self = [super init];
    if (self) {
        _container = sceneContainerNode;
        _wingedEdgesShape = [[ISShape alloc] init];
        __weak Isgl3dNode* weakContainer = _container;
        __weak ISShape* weakWingedEdgeShape = _wingedEdgesShape;
        _wingedEdgesShape.faceFormed = ^(ISFace* face) {
            NSEnumerator* verticesInFace = [face vertexPositionEnumerator];
            ...
            ...
            [weakContainer addChild:sphereNode];
            ...
        }
    }
}
```

Code 4-13

Blocks, or closures have many different usages. Some as multithreading with blocks (see chapter 4.5.3) is further described in the thesis.

4.5 Furniture

Moving furniture objects around the scene is an important functionality in the application. After furniture objects are added to the scene from the library they are added to the scene graph and rendered.

From the interaction point of view, selecting and moving objects is straightforward. The affordance to tap an object and to move it with a pan is obvious. Rotation does not have a clear affordance, but it's a common gesture used in iOS apps.

Collision detection helps placing the objects in the scene. The collisions detection between the objects is simplified by the restriction placed on rotating them only about the y-axis. This way, when panning an object, the distances of the four base point of its bounding box are computed to every other object in the scene. When the value crosses a threshold, the object stays in place instead of continuing with the pan.

The furniture loading was originally done with the Assimp library. A better way, directly from the Imagination Technologies, the GPU vendor, is the PowerVR library [17]. The library is written in C++, and is able to load binary data in the .pod file format. I used the set of tools from the PowerVR SDK to convert models to the .pod format.

4.5.1 Navigating the furniture library

To navigate an item library efficiently I designed a set of table views, which show up in a popover after tapping the “Add Furniture” button in the application menu (Figure 4-18). The tables form a tree hierarchy, which can be traversed by entering tapping the respective category. The categories can have subcategories, as the whole path of traversal is recorded; until a level with items is reached. The items at then add the corresponding model to the scene (Figure 4-19).

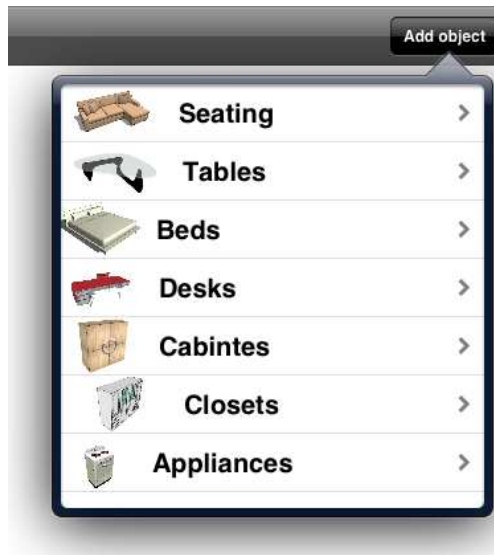


Figure 4-18

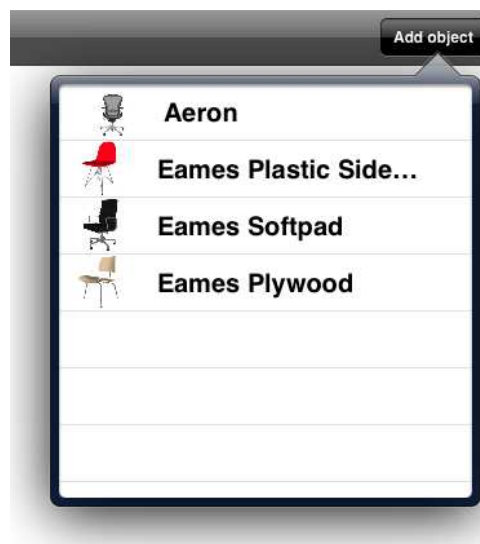


Figure 4-19

4.5.2 Table views

Table views allow for complex behavior in iOS applications. There are highly customizable and can be found in nearly all application.

The two ways to extend the default table view is by inheriting from UITableViewController and by implementing the

UITableViewDelegateProtocol, which is done in the ISItemLibraryTableViewController class (Figure 4-20).

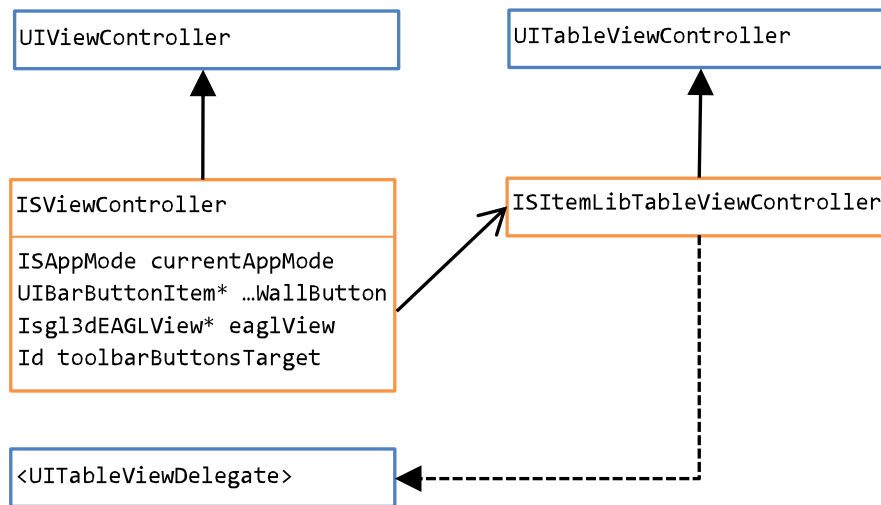


Figure 4-20

The tables are made up by cells – `UITableViewCell` (Figure 4-21), which have their own subviews (Figure 4-22). These subviews are modified in order to represent different places in the table hierarchy, e.g. the categories have a chevron icon in their accessory view.

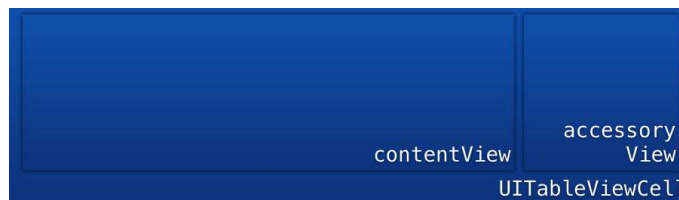


Figure 4-21



Figure 4-22

Each time a table view is displayed, or scrolled, it queries its data source (`ISItemLibTableController`) for the cells it should display. At this point, the 3D models corresponding to the cells shown in the table view are preloaded to save time after selecting them from the list.

4.5.3 Blocks for threads/loading models

The preloading of models is done asynchronously with the utilization of the Grand Central Dispatch (GCD) system [16]. This allows concurrent execution of code on multicore hardware. The GCD is accessible through a C language API.

To keep the UI responsive, blocking operations and other expensive activity is done in separate threads. The time it takes to load the 15 test models in the application is in order of seconds. If done in the main thread, it freezes the UI for the time of loading.

The application checks whether the meshes for the current objects have to be loaded, if so, it queues the model loading code (Code 4-14). There are two blocks (beginning with the `^` sign) and two queues in the example. The blocks are put on a queue and executed asynchronously. When the outer block – the one sent to the `modelLoadQueue` – reaches the point, where it has to interact with the user interface it simply adds the code that should be executed to the queue of the main thread. This blocks the user interface for a short while, but the main part of the computation is finished by that time.

With this approach, we can think of the code in the example as a continuous sequence, even if it is executed asynchronously.

```
dispatch_queue_t modelLoadQueue = dispatch_queue_create("model loader", NULL);
dispatch_async(modelLoadQueue, ^{
    Isgl3dPODImporter* podImporter = [Isgl3dPODImporter
podImporterWithResource: model];
    [podImporter printPODInfo];
    Isgl3dMeshNode* modelNode = [[Isgl3dMeshNode alloc] init];
    // Add all meshes in POD to the node
    [podImporter addMeshesToScene:modelNode];
    [modelNode setScale:0.1f];
    dispatch_async(dispatch_get_main_queue(), ^{
        [_container addChild:modelNode];
        for (Isgl3dMeshNode* subMesh in [modelNode children]) {
            subMesh.interactive = YES;
            [subMesh addEvent3DListener:self
method:@selector(furnitureTouched:) forEventType:TOUCH_EVENT];
            UITapGestureRecognizer* furnitureTap = [[UITapGestureRecognizer
alloc] initWithTarget:self action:@selector(furnitureTapped:)];
            [subMesh addGestureRecognizer:furnitureTap];
            UIPanGestureRecognizer* furniturePan = [[UIPanGestureRecognizer
alloc] initWithTarget:self action:@selector(furniturePanned:)];
            [subMesh addGestureRecognizer:furniturePan];
        }
    });
});
dispatch_release(modelLoadQueue);
```

Code 4-14

4.6 Mathematical functions and unit testing

There were a few mathematical functions I had to implement in order to compute different transformations between bases object intersections and object distances.

One of the implemented functions was `gluUnProject`[18], which maps window coordinates to object coordinates. It is necessary in order to get the correct positions of the gestures in the scene. For this computation, the function needs to know the model view transformation matrix of the scene, the camera model view transformation, the projection matrix and the viewport properties.

In older OpenGL implementations a similar function usually used the depth component of the window coordinates from the depth buffer, which is not accessible in OpenGL ES 2.0. Instead, a ray is created based on the positions of the window coordinates extended by the z-coordinate of the near and far clipping planes.

The other utility functions can intersect this ray with different objects. Most of the times, the ray is intersected with the xz plane, in which the wall nodes are being drawn.

The functions are written in pure C. There is no need to maintain a state – they always output the same result for the same input. This makes them easy to test with the Xcode unit-testing environment. The testing environment is based on the open-source SenTestingKit framework, which provides a set of classes and command-line tools to check the correct functionality of a unit.

5 Evaluation

5.1 Research Goals

The research goals are based on the need to identify how different people interact and understand the interior design application.

The goal main goal is to collect qualitative data about the following product issues:

- To understand the way people navigate in a scene
- To understand the way people combine gestures
- Focus on how differently do people move their finger during gestures
- To understand how people discover affordances of objects in the scene
- To observe how people react to different views and changing of tools
- To understand how users think about what they're trying to do, assumptions
- To identify other pains the users have using the product

5.2 Task Analysis/Usability Test

The research was conducted as 1 on 1 meeting focused on how people are cable of achieving basic tasks with the application. The inquiry was conducted in person in the person's home environment [19].

- The target audience were the users of touchscreen devices, who own or are likely to use a tablet
- The demographics makeup was 5 users aged 20 – 35, with different experience levels, 1 was a long-time iPad user
- The tasks they regularly do included using their tablets/phones for browsing the web, sharing pictures, emails, etc.

The pre-interview screener was based on the basic requirements, that the users should not be professionals in fields like HCI, design, or programming. In order to focus on their direct experience with the application and not on suggestions based on previous knowledge.

The test had the following informal structure:

- Try to draw a room
- Try to add a piece of furniture
- Observe whether the users are trying to explore the objects (by tapping, etc.)
- Remove a previously designed room and add new walls
- Try to get closer to a certain point in the scene

5.3 Findings

Direct suggestions by the users:

- One user found the wall lengths unreadable
- Complained about the lack of “snapping to grid”
- One user had a hard time turning of the wall drawing mode
- Most of the users liked the way of navigating in the scene

Observation:

- The more advanced users automatically assumed that kinetic scrolling is available
- Everyone was able to pan a furniture model, without explicitly asking them to do it
- Two users did not try to rotate a selected furniture item
- Users tried to change object properties by tapping them, the two most advanced users tried a long tap
- One tried to resize the room by dragging the wall
- A users tried to “scroll back” in the furniture library to get back to the categories
- When misplaced a wall, the user tried to remove it by tapping it, which instead of deleting resulted in drawing another wall
- Only tapped the scene in draw mode, instead of trying to continue the pan
- Users tended to be explorative and curious, they tried to interact with objects
- One user tried to pan the view when drawing walls, which draw another wall instead
- Users tried to end walls in the middle of other walls
- Users tried to remove wall by tapping on them in the default view

5.4 Design suggestions

The most important design suggestions are:

- Create a simple image showing how to add walls
- Make the wall lengths more visible
- Add visual clues to show that the furniture can be rotated
- Fix the functionality to start dragging walls “from” other walls
- Add popovers to walls, compare with the delete mode

All but the first suggestion has been implemented in an updated version of the application.

6 Similarity search

This chapter discusses the problem of similarity search and shape retrieval. It begins with a short overview of the state of the art techniques and introduces the necessary theoretical tools to understand the domain of the problem. The rest of the chapter is dedicated to the implementation of a specific similarity search algorithm and the toolchain involved in its implementation.

6.1 The similarity search problem

The development of modeling techniques and the growth of the Internet led to an explosion of the number of available 3D models. This meant a widespread availability of 3D models for various visualization purposes. This new found scale required the development of 3D shape retrieval systems that, given a query object, retrieve similar 3D objects.

A real life example of similarity search can be found in Trimble/Google 3D Warehouse. It probably ignores any written description of the models and relies solely on geometrical similarity, which can sometime yield irrelevant results (Figure 6-1). One can hardly think of a situation where stating that a window, a motherboard, and a skatepark are similar is of any use. However, the geometrical similarity is not the main way of finding objects in such databases, a searchbar, where the user enters the description of the requested object is the most common way.

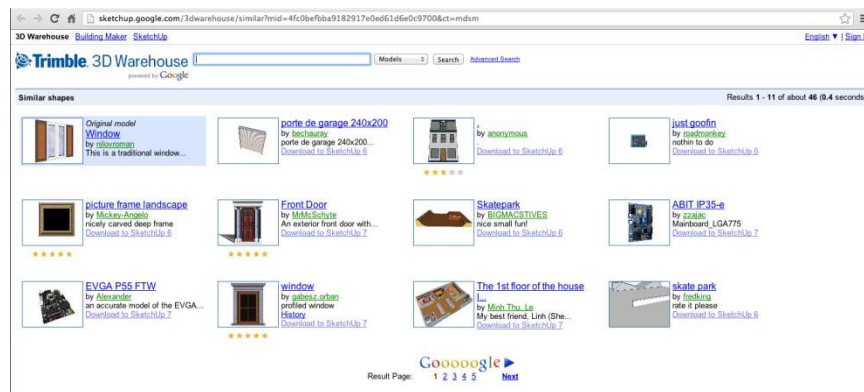


Figure 6-1

6.1.1 Shape retrieval/categorization overview

Shape retrieval techniques can be categorized based on different criteria. The techniques are often combinations of different approaches with overlapping category boundaries. One of the distinctions helpful in navigating the space of techniques is studying the different elements of the similarity search systems. A system would usually comprise a query interface, a similarity computation technique, and a model representation.

A good overview was published by J. Tangelder and R. Veltkamp [21], where different techniques were discussed and evaluated based on the following criteria:

- shape representation requirements

- properties of dissimilarity measures
- efficiency
- discrimination abilities
- ability to perform partial matching
- robustness
- necessity of pose normalization

For the needs of this thesis the requirements for shape retrieval are specifically constrained by the domain of furniture and interior decoration objects. This helps in choosing the right method of similarity search.

The shape representation of furniture models is nearly exclusively a polygonal mesh; the structure is of a less importance than the visual appearance. With this in mind, it is safe to discard all the techniques based on volumetric data representation.

The properties of dissimilarity measures are a set of mathematical definitions, from which the most interesting one deals with the symmetry relation. Symmetry is not always wanted. Indeed, human perception does not always find that shape x is equally similar to shape y , as y is to x . In particular, a variant x of prototype y , is often found to be more similar to y than vice versa [21].

Global vs. partial matching is an important consideration (Figure 6-2). In global matching, the similarity between objects is examined on the whole. In partial matching, components of the objects can be selected, and the similarity (or dissimilarity) measure is refined by them.

Shape distributions distinguish models in broad categories very well: aircraft, boats, people, animals, etc. However, they often perform poorly when having to discriminate between shapes that have similar gross shape properties but different detailed shape properties.

In contrast to global shape matching, partial matching finds a shape of which a part is similar to a part of another shape. Partial matching can be applied if 3D shape models are not complete, e.g. for objects obtained by laser scanning from one or two directions only. Another application is the search inside “3D scenes” containing an instance of the query object.

The initial use of the partial matching techniques was based on the premise that the user draws a 2D image of what he wanted to search for. This is a lengthy and cumbersome task. It is hard to imagine such an interface in a real world application.

Some recent research shows very promising results for discriminating objects based on selecting parts of the original objects, which the user likes/does not like [32]. The amount of improvement over global methods in a specific domain is not clear.

Transformational invariance and robustness are important properties from both the users and implementation points of view. The criterion of robustness means that small changes

in a shape should result in small changes in the shape descriptor. On the other hand, if large changes in the shape of the object result in very small changes in the shape descriptor, then the shape descriptor is considered not sensitive enough. Poor sensitivity will lead to poor discriminative abilities.

In the absence of prior knowledge, 3D models have arbitrary scale, orientation, and position in the 3D space. Because not all dissimilarity measures are invariant under scale, translation, or rotation, one or more of the normalization may be necessary.

To successfully accomplish a task, it is essential to define what the final result should look like. Translated to the case of similarity search, what is considered to be a good solution? How can the quality of two solutions be compared?

The effectiveness of shape retrieval is often compared by measures such as precision, recall, bull's eye percentage, k-th tier, ROC curve, specificity, total performance, and relative error. Such performance measures depend on the chosen query, embedding database, and chosen ground truth. They are not an inherent property of the retrieval method, so it is difficult to give exact, objective numbers. [21]

For the purpose of this work, it seems to be more appropriate to approach the definition of the goal from the user's perspective. A good result would provide the user with a number of alternatives to the currently selected model, which the user might prefer.

It is also useful to understand the fact, that the current benchmarks like the University of Konstanz test set or the Princeton Shape Benchmark work with models spanning 90 classes with the total of only 1814 models. If the models are distinct enough, the similarity search can be thought of as a categorization problem, where similar objects make up a category.

An algorithm that would perform well in these tests might not be suitable for a domain specific application like furniture similarity, where objects have much smaller variance, e.g. an airplane vs. a bookshelf and a fridge vs. a bookshelf.

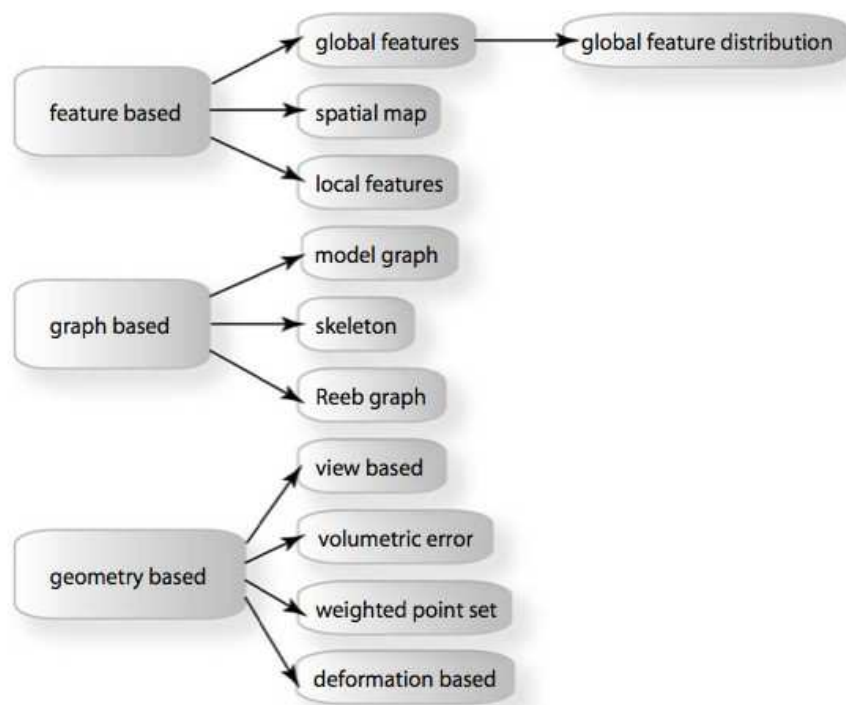


Figure 6-2

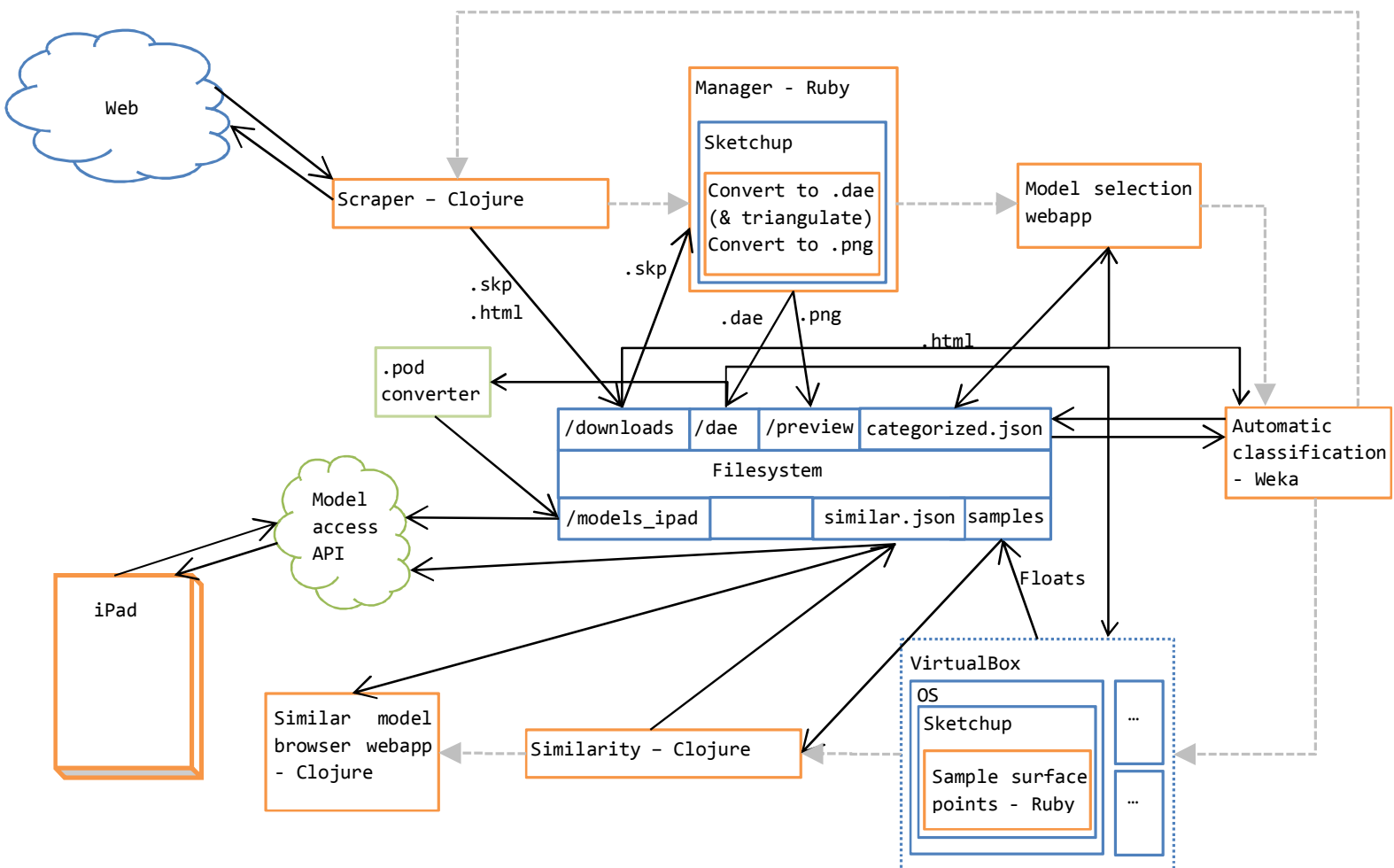


Figure 6-3

The overview of the implemented system can be found in Figure 6-3. Some details were omitted for clarity. The building blocks were created to be as separate and independent as possible. Instead of a complex messaging protocol or API calls, they communicate with each other through the filesystem. One stage stores its output; if the other stage finds unprocessed files, it begins processing them. The filesystem is very effective for storing files and keeping the system decoupled. Some operations like listing the files in a directory with thousands of files are manually cached in a json map for performance reasons. The orange boxes in the diagram were implemented by me, the blue ones are existing components, and the green ones have to be implemented. The gray lines denote the abstract information flow; the services do not call each other directly.

The steps performed by the process are the following:

1. File scraping
2. Preview generation and file format transformation
3. Manual furniture selection
4. Text based classification
5. Geometry based classification

The implemented process can be broken down into 5 steps. The order of these steps is not strictly linear and can be modified. Also, the granularity does not follow any strict rules. The functionality of the blocks could be merged or split based on performance and implementation requirements.

6.2.1 File download/scraping

Input: [Hyperlink to a web page with a 3D model](#)

Output: 3D model files, html files with model descriptions

To obtain a large enough sample of 3D models, I have written a scraper for traversing web pages and downloading available models.

The scraper was first seeded with a webpage of a 3D model. This page included a description, a download link, and links to further models. The program proceeded with following the hyperlinks in a BFS manner and downloaded the models. It is worth noting that most of the downloaded models were not furniture. The online model repositories contain general models and are not domain specific. Ways to deal with this limitation are discussed later in this chapter.

The scraper successfully downloaded 80 000 models and their descriptions (with an estimated 1/10 being furniture). However some of the files were invalid; their headers had to be checked before using them in the next stage.

This approach also means that the models come from different authors and in a variety of formats, quality, and size. This has to be accommodated for during the similarity computation by either using a more robust approach or normalizing the objects.

The program itself is written in Clojure.

6.2.2 Preview generation and file format transformation

Input: Downloaded 3D files

Output: 2D png previews, processed 3D models in Collada file format

As with all the other parts of the work, the architecture described in this part is not the first iteration. It is a snapshot of the latest state. The architecture usually evolved from doing things manually through simple unstable scripts to a reasonably fast and stable structure.

The block described in this chapter is responsible for two operations, which are independent. Putting them together makes sense for performance reasons.

As mentioned earlier, the 3D files come from different authors and their approaches to modeling vary. The files are therefore transformed into a common format – Collada (.dae), which has a good support among 3D modeling tool and is supported by the tools developed by Imagination Technologies. These can produce a dense binary representation (.pod) with performance tuned for the iPad's GPU.

The only modification, besides the format conversion, done to the model is triangulation. It is required by both the .pod converter and the similarity search algorithm.

The second action carried out is the creation of a 2D preview of the models.

Both of these operations are done in the environment of Trimble SketchUp (formerly Google SketchUp) modeling tool. It allows for running Ruby scripts through the inbuilt interactive console. The scripts have access to the geometry and topology of the model, as well as other features of SketchUp, to seamlessly integrate as extension modules.

The advantage of running scripts within SketchUp is having an abstraction over models from different authors, although advanced functions like grouping make it harder to accommodate for all the possible representations. It required a lot of trials and errors to make the code work with real world models.

The second advantage is faster development and interactive testing of functions, which can interact with the model and provide realtime feedback.

One of the shortcomings of running SketchUp scripts is the performance, but the speed of development was a strong reason to stick with SketchUp. As shown later in this chapter the performance can be vastly improved by digging deeper in the exposed API and by running multiple instances in parallel.

There are two Ruby scripts involved with this block. The first one takes care of the housekeeping – selecting the files for conversion, launching and restarting SketchUp, checking the state of the conversion, etc. This script can be thought of as a wrapper around SketchUp to create a stable environment for the second script that is launched inside of the application with access to the model.

6.2.3 Manual furniture selection

Input: 2D model previews, html model descriptions

Output: An attribute stating whether the model is a furniture item

The aim of this step is to separate the furniture models from the all the other models. It is needed for the similarity search algorithm, and as a learning step for the text based supervised learning.

If the supervised learning is successful, there is no need to perform this step more then once (or no need to do it at all, if an unsupervised algorithm produces a viable result).

In order to make the classification of a large number of files possible, I made an application that presents the models as a webpage and processes the user input (Figure 6-4).

The webpage displays the models in a concise table, with their tags, and radio buttons for selecting the desired models. The selection can be submitted as a form, which is saved as a json map from the model id to its category (“furniture”, “not furniture”).

It took me approximately 2 hours to manually categorize 1500 models with this interface. It was also written in Clojure, with handy libraries for interfacing with the Jetty Http server, and html generation [23].

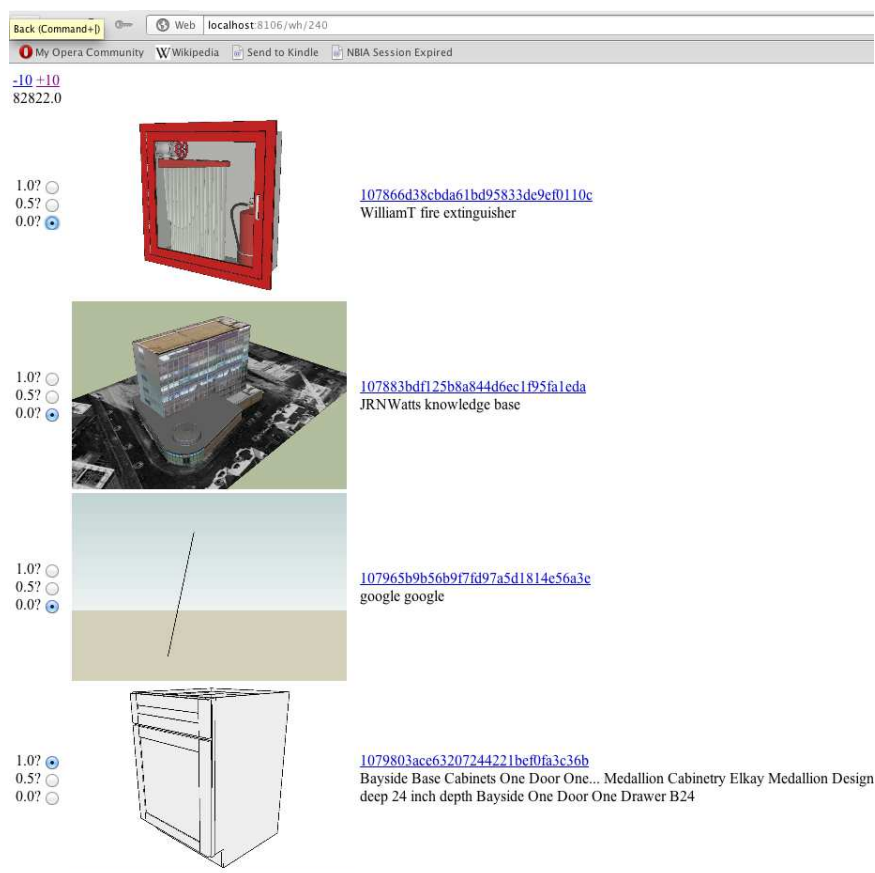


Figure 6-4

6.2.4 Text based classification

Input: html pages for models, manually classified learning set

Output: Classification of the whole dataset

The reason for employing this step is twofold. First of all, downloading gets worse over time. By predicting whether a webpage contains a furniture model time and bandwidth could be saved. The html page is orders of magnitude smaller than the whole model.

The second reason to classify the models based on their written description is to remove the unnecessary models from the further steps of the similarity search. These models would get picked up as noise by the algorithm. The filtering allows the geometric similarity computation to be less robust as it does not have to accommodate for unwanted models.

Currently, the models are classified based on their tags and put into two categories – “furniture” and “non furniture”.

The models downloaded from the Web have textual description and a manually assigned category. One can assume that the higher quality models are properly described and tagged.

The classification, or learning, consists of three steps:

- preparing the learning data
- preparing the unclassified data
- running an appropriate machine learning algorithm on the data

The first step was discussed in the previous chapter. The second and third depend on the choice of the machine learning architecture. One can program all the necessary tools for himself, use a machine learning library, or use a more complex tool with a data exploration interface.

I decided to use Weka [31] for this stage. It comes with a GUI and a large library of the most common machine algorithms and data filters. It allows interactive exploration of data and testing of different parameters.

Weka can also be used programmatically from Java. However, the main benefit is the interactive data exploration interface. I have written an application in Clojure that transforms all the html pages with the descriptions to a single arff file (Weka’s default file format) and continued to work with it inside of Weka.

Validating the output of standard machine learning algorithms is more straightforward than it is in the case of geometric similarity. We know how a good output should look like. The models should be put into the right categories. Test data can be prepared to validate the classifiers.

However, stating that a good algorithm is the one that classifies the most data from test set into the right category is a very sloppy and incorrect definition. It does not tell us anything

about how the learned function will behave outside of the training data. To be accurate, one has to use a probabilistic argument about how the learned function will behave “in the wild” [27].

Again, it can be misleading to say that a function with 90% classification precision on the test data is better than a function with 85% precision. It could simply be a consequence of wrong generalization or overfitting. Because of these reasons, I have created another webapp that displays the output of the algorithms (Figure 6-5) to manually assess the output.



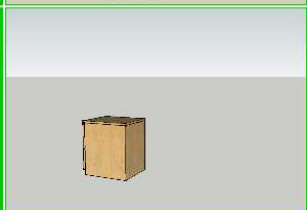
	0.998 10b8ba19f579d38a719f5da46e0327a1 Randall Crane knowledge base 130 main street art deco evansville indiana
	0.368 10bc7b5ae57c52ffd853285e74aba51b Green Thumb Model Awards Costy
	0.322 10bdcf2d0da9aac68caf5ada882597a IKEA furniture Noah Storage malm ikea drawer chest desk
	0.978 10b8a05283da8532290b18c50731db6 Marian87 Ferrari 550 barchetta fast car Marian

Figure 6-5

After importing the arff data to Weka, it has to be preprocessed in order to feed it into the machine learning algorithms [25].

In preprocessing, the string data from the tags and description extracted from html pages are tokenized to a vector containing single words and their frequencies. During the process the words are stemmed. In Weka terminology, each word becomes an attribute [28]. The last attribute is the class of the object (“furniture”, “non furniture”).

To prevent overfitting, the number of words in the vector is reduced to 1000 (Fragment 6-1), which yields better results than using 10 000 words (Fragment 6-2) for the Bayes classifiers. It does not affect the decision trees based classifiers (Fragment 6-3).

The manually categorized data is split into a training set and a set of validation instances. The classification can be statistically assessed by the properties of true positives (TP), false positives (FP), true negatives (TN), false negatives (FN), or ROC area.


```

...
| | | | | | | | | display <= 0
| | | | | | | | | | ikea <= 0: 0.0 (19.0/3.0)
| | | | | | | | | | ikea > 0: 1.0 (2.0)
| | | | | | | | | display > 0: 1.0 (2.0)
...
...
| | | | | kitchen > 0
| | | | | | in <= 0
| | | | | | | dining <= 0
| | | | | | | | with <= 0: 1.0 (18.0/1.0)
| | | | | | | | with > 0
| | | | | | | | | three <= 0: 0.0 (2.0)
| | | | | | | | | three > 0: 1.0 (2.0)
| | | | | | | dining > 0: 0.0 (2.0)
| | | | | | in > 0: 0.0 (4.0)
| | | | table > 0
| | | | | be <= 0
| | | | | | models <= 0: 1.0 (22.0/3.0)
| | | | | | models > 0: 0.0 (2.0)
| | | | | be > 0: 0.0 (3.0)
...
...
Number of Leaves :      56

Size of the tree :      111

```

Fragment 6-3

A row in the decision tree (Fragment 6-3) can be interpreted as follows, the first number is the total number of instances (weight of instances) reaching the leaf. The second number is the number (weight) of those instances that are misclassified.

```

Instances:    1573
Attributes:   2058
[list of attributes omitted]
Test mode:split 66.0% train, remainder test

=== Classifier model (full training set) ===

The independent probability of a class
-----
0.0      0.7815873015873016
1.0      0.21841269841269842

The probability of a word given the class
-----
                                0.0 <- not furniture    1.0 <- furniture
...
...
...
fun          3.489914148111958E-4    9.828975820719488E-5
furniture    5.583862636979128E-4    0.005209357184981327
fut          3.140922733300763E-4    9.828975820719488E-5
future       6.630836881412717E-4    9.828975820719488E-5
game         7.328819711035107E-4    1.9657951641438977E-4

```

garage	6.281845466601522E-4	9.828975820719488E-5
...		
...		
...		
guitar	3.8389055629231534E-4	9.828975820719488E-5
gun	0.0017100579325748581	9.828975820719488E-5
...		
...		
...		
office	9.422768199902285E-4	0.001179477098486338
Correctly Classified Instances	482	90.0935 %
Incorrectly Classified Instances	53	9.9065 %

Fragment 6-4

Fragment 6-4 shows two columns, from which the conditional probability of a class given a word can be derived using Bayes' Theorem [33]. We know the apriori probability of being a furniture and can derive the posterior probability after observing (or not observing) a word in the input vector for the instance being classified.



Figure 6-6

Further improvements could include clustering, or the use of multiple categories, not just a binary output. An extension to word combinations could rule out more models in a single scene – we do not want the tags “chair” and “table” to appear together in a description.

6.2.5 Geometric descriptor

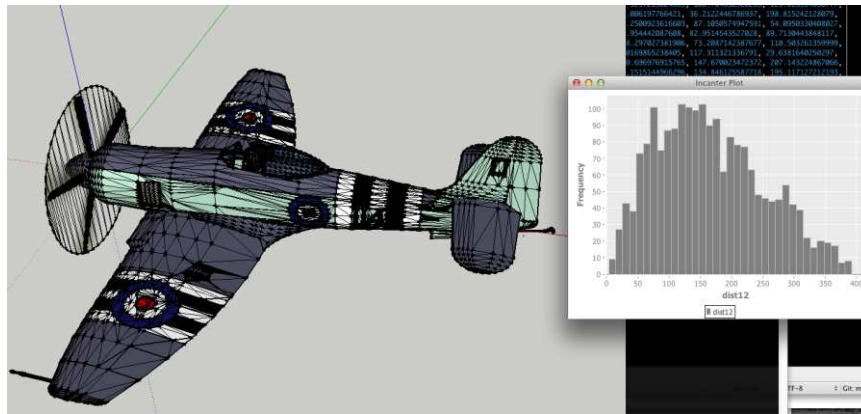


Figure 6-7

Input: Triangulated 3D (furniture) models in Collada file format

Output: Similarity coefficient for pairs of models

The geometrical similarity computation is based on the “Shape Distributions” method [30]. According to the classification introduced earlier in this chapter it is a global method, more specifically a global feature distribution method (Figure 6-2).

The algorithm creates a probability distribution of a feature for each model. These distributions are subsequently compared for each pair of models. The ones with the smallest distance between their respective distributions are considered to be similar.

The distributions are histograms built from values of functions over the points making up the model. These functions are geometric properties – shape functions – like the distance of two points, the distance of three points, the angles between points, etc. The histogram tells us, how often a certain value of the function is represented in the model.

The cited paper considers all the aforementioned functions to be equally good for this task. I used the $d2$ function – distance of two points, as it was recommended by the authors of the paper.

The method consists of the following steps:

1. Sampling of point pairs
2. Computation of the Euclidean distance between the pairs
3. Building a histogram of distances from all the sampled data
4. Comparison of histograms of different models

The first two steps on the list are carried out within SketchUp as Ruby scripts; the remaining two are done by standalone Clojure programs.

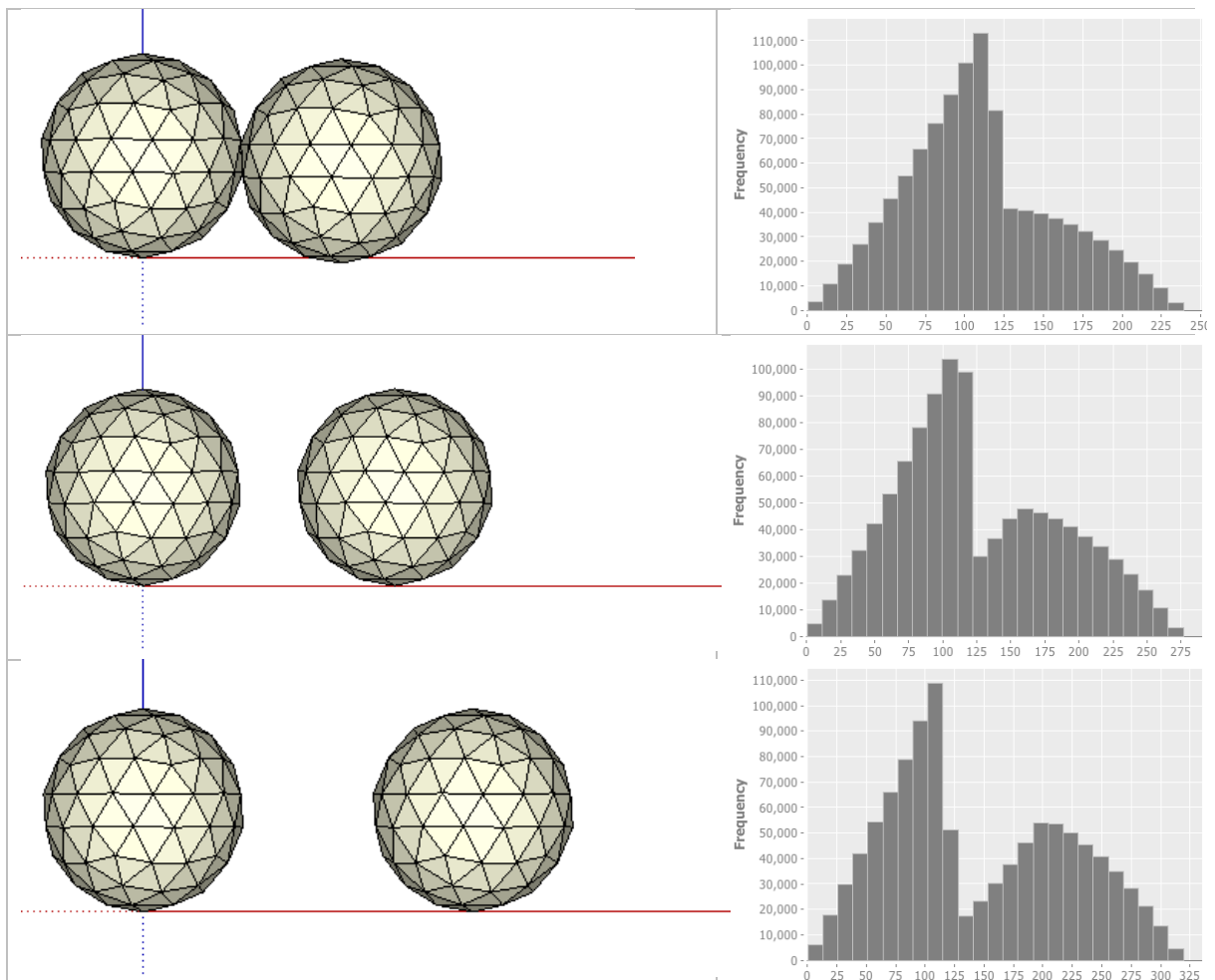
To get the points from the surface a stochastic method is employed. The simplest approach would be to sample vertices of the 3D model directly; the resulting shape distributions would be biased and sensitive to changes in tessellation.

Using the vertices of 3D models was the approach I used at first, because it did not require triangulation of polygons. The results were good enough to build the other parts of the system and refine the sampling later. The speed of triangulation in SketchUp had to be solved. It was too slow to do it in the Ruby script. A better approach was performing the triangulation in one of the previous processing stages.

In the latest version the shape functions are sampled from random points on the surface of the 3D models. The choice of unbiased random points with respect to the surface area of a polygonal model proceeds as follows. First, the algorithm iterates through all triangles. Then for each triangle, its area is computed and stored in an array along with the cumulative area of triangles visited so far. Next, a triangle is selected with probability proportional to its area by generating a random number between 0 and the total cumulative area. This is used to perform a binary search on the array of cumulative areas.

For each selected triangle with vertices (A, B, C), a point on its surface is constructed by generating two random numbers, r_1 and r_2 , between 0 and 1, and evaluating the following equation:

$$P = (1 - \sqrt{r_1})A + \sqrt{r_1}(1 - r_2)B + \sqrt{r_1}r_2C \quad [30]$$



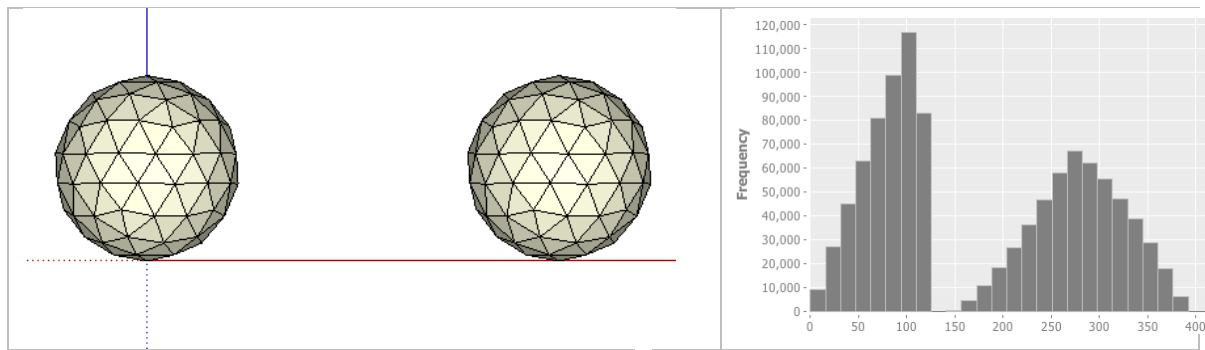


Figure 6-8

With this approach, it's equally likely for each point on the surface to be selected. From each model 1024^2 samples are taken.

For each pair of points, their Euclidean distance is computed. The resulting distances are stored in a file as an array of float values, so they can be processed outside of SketchUp from this point on.

The stored float distance values are again processed in Clojure. The samples from the shape distribution are used to construct a histogram by counting how many samples fall into each of the 1024 fixed sized bins [22]. These settings ensure low enough variance and high enough resolution to be useful for the initial experiments.

This number is a recommendation from the article, it can be adjusted to get a higher resolution distribution, or lowered to make the shape descriptor more insensitive to noise and small extra features, and robust against arbitrary topological changes.

The examples of such histograms can be seen in Figure 6-7 for the Spitfire, and in Figure 6-8 for two spheres with increasing distances. The left part of the histogram in the case of the spheres is the distribution of distances within individual spheres, whereas the “moving part” represents the growth of the distances sampled from both spheres.

After binning the data the histograms are normalized by the number of total samples. These new histograms can be now interpreted as frequencies.

There are multiple techniques of comparing two histograms. I used the Chi-squared test [34]. Alternatively a statistical method not requiring binning, such as ANOVA, could have been used.

Thanks to the stochastic sampling, binning, and histogram normalization there is no need to do pose normalization. At each steps, the values are relative. This means that the descriptor is invariant to scale, rotation, and translation. It is obviously advantageous for rigid transformations. The models are often placed at different locations in the scene. The property of scale invariance is questionable. It would be undesired for data with specified dimensions. The gathered furniture models do not have their dimensions specified; therefore they should not be discriminated based on their scale.

As mentioned earlier, the first two steps are carried out in the SketchUp console. This way, the computation becomes strongly single threaded and can't utilize multiple processors.

As the computations are independent the input batch can easily be partitioned into independent chunks. Each chunk is assigned to a virtual machine running SketchUp. After setting up the partitioning and sharing, it was easy to scale this architecture to achieve 100% utilization of my development machine by running 4 VirtualBox instances. The speed improvement was linear with the number of machines added.

The performance of the geometric classification with the current architecture is roughly 100s of models per hour. The SketchUp computation takes up most of the time. The histogram computation and comparison is done in a matter of minutes. This is a good result considering the need to compare the histograms for all the combinations. As the processing of histograms is independent of each other the parallel mapping function – pmap – in Clojure can be exploited [26]. It is well suited for multiprocessor machines as it creates a future/thread for each of the computationally expensive operations.



Figure 6-9

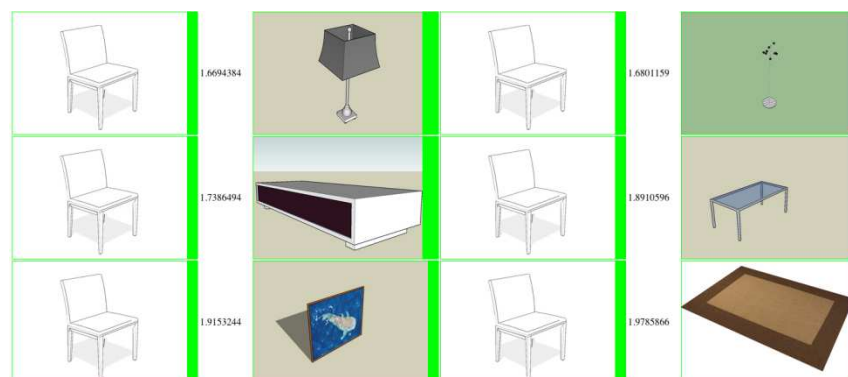


Figure 6-10

The output values of Chi-squared tests are usually compared by taking the number of degrees of freedom of the input set and comparing them to the Chi-square distribution. In the case of picking the most similar models, it is not necessary. The similar models can be sorted in ascending order from 0 (being the most similar - Figure 6-9) up until the most dissimilar (Figure 6-10). The only caveat is that these numbers can't be compared for

different models. They only make sense as a relative comparison to the base model (the white chair in the figures).

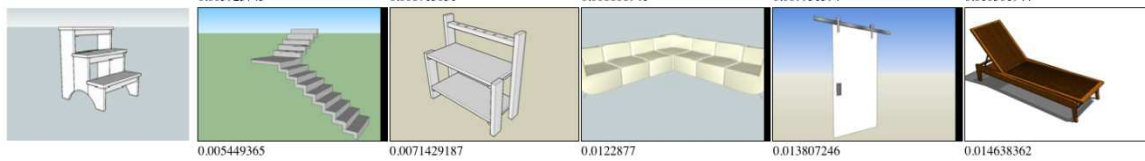


Figure 6-11

The similarity search would improve even more by adding more models. From the users perspective we are interested in the most similar 5 or 10 objects to the current one. I.e. we do not need the descriptor to match all the similar chairs; only to have enough similar ones to fill up the 5 closest spots (Figure 6-11). A direct consequence of the observation that more models are better is the rise in absolute number of “noisy models”. Noisy models are mainly non furniture models that have a very similar shape distribution as a valid furniture model. To mitigate the decay of similar model search after expanding the number of models it is important to reduce the relative number of “noisy models”. This number maps directly to the percentage of false positives in the text based classification. Thus by reducing the number of the, easy to measure, false positives in the previous step, the output of the geometric classification will improve.



Figure 6-12

6.2.6 Possible improvements and conclusion

The current implementation can be improved along the dimensions of accuracy and performance.

A close look at the results of the classification reveals that the geometric proximity of the objects recognized as similar is high.

Unfortunately, man made objects have many common features. A fridge and a wardrobe are geometrically very similar shapes. Some visual distinction can be made by examining their color and texture. The same is true for a sofa and a coffee table, where a large

horizontal area is connected to 4 smaller legs. Furniture is built of a finite set of building blocks that can be found in many furniture items with different functionality.



Figure 6-13

Nevertheless, the geometric classification works well inside of furniture categories. When all but the furniture belonging to a single category is considered, the ordering seems to be reasonable (Figure 6-13 and Figure 6-12).

It is very likely that an extension of the text based classification could perform this crucial improvement. It could be achieved by expanding the classification from the categories of “furniture” and “non furniture” to categories like “chairs”, “beds”, “sinks”, etc.

Either clusters created with unsupervised methods could be manually labeled, or a supervised method with a modified training set could be used.

The use of textual description of the models would then allow discriminating glass doors from windows, which are visually very similar. Using tags, the description is straightforward for these cases.

Performance wise, the lowest hanging fruit for optimization is replacing the launching of scripts inside of SketchUp with direct access to the models. The current setup is still useful while improving the accuracy of the process.

The performance optimizations are vast; ranging from running transformations in parallel on multiple machines to utilizing GPUs. As an example, after removing a few hotspots with profiling the current setup improved from batch processing of 100 correspondences in 1 day to the current state of roughly 100 000 correspondences.

7 Conclusion

The thesis was focused on building an application (Figure 7-1) for the iPad. Besides trying to create a useful application, one of the aims of this work was to get a feel for the platform and understand the Objective-C language. The only way to accomplish these goals is to write programs, read other people's programs and learn from the documentation.



Figure 7-1

I analyzed the current state of the interior design applications, and successfully implemented and tested a new application that tried to address the issues identified by the analysis.

During the development I tried different approaches to solve software design issues. Most of them did not lead to desired results, but some did. These can be used as the foundations for further development.

During the development I explored at least a few edges of the platform, and understood some of the possibilities it brings about. The tablet computer offers an exciting new way of user interaction, with many usage scenarios and patterns yet to be discovered. Many user experience aspects of multitouch interfaces are being defined right now. They are often contradictory and will require further investigation by HCI researchers. The raw computing power, graphics processing capabilities, and high resolution display make the iPad a tool well suited for applications in visualization and novel computer games.

The category of tablet devices already does some things better than the desktop computers, which still represent the default way of thinking about computing today. By removing a layer of separation in the form of a mouse and a keyboard, web browsing, video calls, or reading news becomes ever more convenient.

Tablets and the iPad should not be dismissed as toys by simply looking for features that they clearly lack when compared to desktops. Their main use is not authoring text. Nevertheless, there is a large set of applications that can be taken from the desktop paradigm and transferred to tablets to provide a better user experience. Especially applications where text input is not a core part of the job being done.

To contradict the allegations of being a toy, new software has to be built. The question then becomes whether it can be built, i.e. is the underlying hardware and software platform capable to provide the resources needed to build it.

After spending months with the platform I must conclude, that it does. The software part offers state of the art features, and the hardware is powerful enough.

Considering the tablet computer's capabilities and its potential of redefining the user experience with computing, we should stop thinking about emulating desktop applications and trying to do something simpler on a tablet. We should think of how tablets can better achieve various goals. Much like the PC was more than just an expensive, hard-to-use typewriter (it was a whole new thing that just happened to have some typewriter features), the tablet is much more than a touchscreen device for viewing family photos [20].

8 Bibliography/references

- [1] P. Graham, “Taste for Makers.” [Online]. Available: <http://www.paulgraham.com/taste.html>. [Accessed: 11-May-2012].
- [2] M. Andreessen, “Pmarchive.” [Online]. Available: http://pmarchive.com/three_kinds_of_platforms_you_meet_on_the_internet. [Accessed: 11-May-2012].
- [3] B. Shneiderman and C. Plaisant, *Designing the user interface : strategies for effective human-computer interaction*. Boston: Addison-Wesley, 2010.
- [4] “Learning Objective-C: A Primer.” [Online]. Available: http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/_index.html. [Accessed: 11-May-2012].
- [5] “iOS Human Interface Guidelines: Introduction.” [Online]. Available: <http://developer.apple.com/library/ios/#DOCUMENTATION/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html>. [Accessed: 11-May-2012].
- [6] “Event Handling Guide for iOS: Gesture Recognizers.” [Online]. Available: <http://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/GestureRecognizers/GestureRecognizers.html>. [Accessed: 11-May-2012].
- [7] “Open Asset Import Library.” [Online]. Available: <http://assimp.sourceforge.net/>. [Accessed: 11-May-2012].
- [8] “iSGL3D.” [Online]. Available: <http://isgl3d.com/>. [Accessed: 11-May-2012].
- [9] “iOS Human Interface Guidelines: User Experience Guidelines.” [Online]. Available: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/UIElementGuidelines/UIElementGuidelines.html#//apple_ref/doc/uid/TP40006556-CH13-SW1. [Accessed: 11-May-2012].
- [10] “Cocoa Fundamentals Guide: Communicating with Objects.” [Online]. Available: <http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/CommunicatingWithObjects/CommunicateWithObjects.html>. [Accessed: 11-May-2012].
- [11] “iPad for Data Viz @ CMU.” [Online]. Available: <http://hci-ipad.org/>. [Accessed: 11-May-2012].
- [12] “iOS Human Interface Guidelines: Human Interface Principles.” [Online]. Available: <http://developer.apple.com/library/ios/#DOCUMENTATION/UserExperience/Conceptual/MobileHIG/Principles/Principles.html>. [Accessed: 11-May-2012].
- [13] J. Arvo, *Graphics gems II*. Boston: Academic Press, 1991.
- [14] “The Objective-C Programming Language: Associative References.” [Online]. Available: <http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocAssociativeReferences.html>. [Accessed: 11-May-2012].
- [15] “Faking instance variables in Objective-C categories with Associative References – Ole Begemann.” [Online]. Available: <http://oleb.net/blog/2011/05/faking-ivars-in-objc-categories-with-associative-references/>. [Accessed: 11-May-2012].
- [16] “Blocks Programming Topics: Conceptual Overview.” [Online]. Available: <http://developer.apple.com/library/ios/#documentation/cocoa/Conceptual/Blocks/Articles/bxOverview.html>. [Accessed: 11-May-2012].
- [17] “PowerVR Insider SDK.” [Online]. Available: <http://www.imgtec.com/powervr/insider/powervr-sdk.asp>. [Accessed: 11-May-2012].

- [18] “gluUnProject(3) Mac OS X Manual Page.” [Online]. Available: <https://developer.apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man3/gluUnProject.3.html>. [Accessed: 11-May-2012].
- [19] M. Kuniavsky, *Observing the user experience : a practitioner’s guide to user research*. San Francisco, CA: Morgan Kaufmann Publishers, 2003.
- [20] P. Buchheit, “Paul Buchheit: Tablet thoughts.” [Online]. Available: <http://paulbuchheit.blogspot.com/2009/12/tablet-thoughts.html>. [Accessed: 11-May-2012].
- [21] J. W. Tangelder and R. C. Veltkamp, “A survey of content based 3D shape retrieval methods,” in *Shape Modeling Applications, 2004. Proceedings*, 2004, pp. 145–156.
- [22] E. Rochester, *Clojure Data Analysis Cookbook*. Packt Publishing, 2013.
- [23] A. Rathore, *Clojure in Action*, 1st ed. Manning Publications, 2011.
- [24] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition*, 3rd ed. Morgan Kaufmann, 2011.
- [25] K. Dent, “Homework 2: Automatic text categorization using Weka -,” *Homework 2: Automatic text categorization using Weka*. [Online]. Available: http://www.cs.columbia.edu/~kdent/classes/4701/hw2_assignment.html. [Accessed: 08-May-2013].
- [26] Fatvat, “JVisualVM and Clojure.” [Online]. Available: <http://www.fatvat.co.uk/2009/05/jvisualvm-and-clojure.html>. [Accessed: 08-May-2013].
- [27] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data*. AMLBook, 2012.
- [28] J. M. G. Hidalgo, “Nihil Obstat: Text Mining in WEKA: Chaining Filters and Classifiers,” *Nihil Obstat*, 29-Jan-2013. .
- [29] J. Atwood, “Podcast #32 « Blog – Stack Exchange,” *StackExchange Blog - Podcast #32*. [Online]. Available: <http://blog.stackoverflow.com/2008/12/podcast-32/>. [Accessed: 08-May-2013].
- [30] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, “Shape distributions,” *ACM Transactions on Graphics (TOG)*, vol. 21, no. 4, pp. 807–832, 2002.
- [31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: an update,” *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
- [32] V. G. Kim, W. Li, N. J. Mitra, S. DiVerdi, and T. Funkhouser, “Exploring collections of 3D models using fuzzy correspondences,” *ACM Trans. Graph.*, vol. 31, no. 4, pp. 54:1–54:11, Jul. 2012.
- [33] E. W. Weisstein, “Bayes’ Theorem -- from Wolfram MathWorld.” [Online]. Available: <http://mathworld.wolfram.com/BayesTheorem.html>. [Accessed: 09-May-2013].
- [34] E. W. Weisstein, “Chi-Squared Test -- from Wolfram MathWorld.” [Online]. Available: <http://mathworld.wolfram.com/Chi-SquaredTest.html>. [Accessed: 09-May-2013].

9 Application screenshots

Default application view (Figure 9-1).

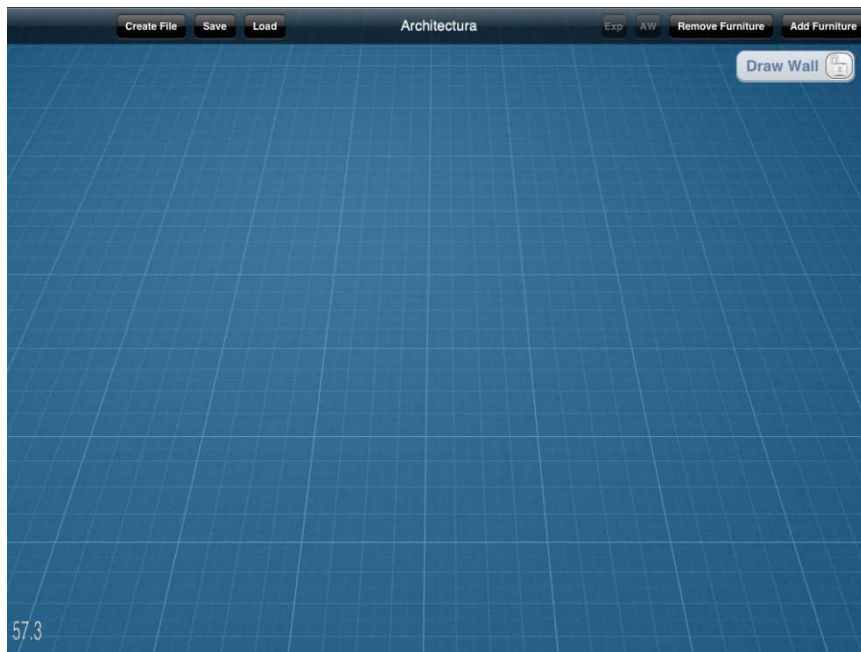


Figure 9-1

Wall drawing mode (Figure 9-2).



Figure 9-2

Drawing a wall, the view is top-down. While drawing, the end snaps to interesting lengths and angles to assist the user. (Figure 9-3)

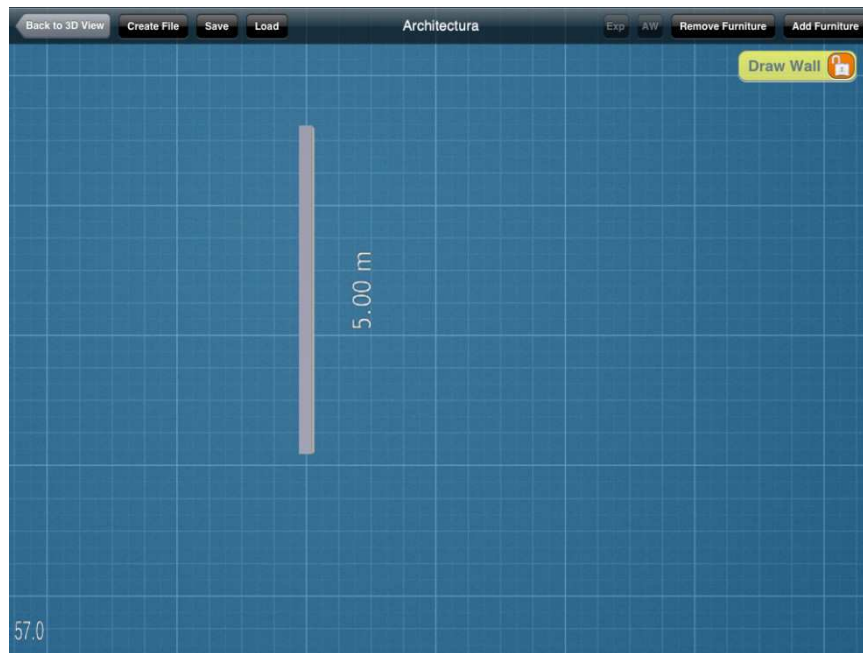


Figure 9-3

Connecting a new wall to the end of an existing one (Figure 9-4).

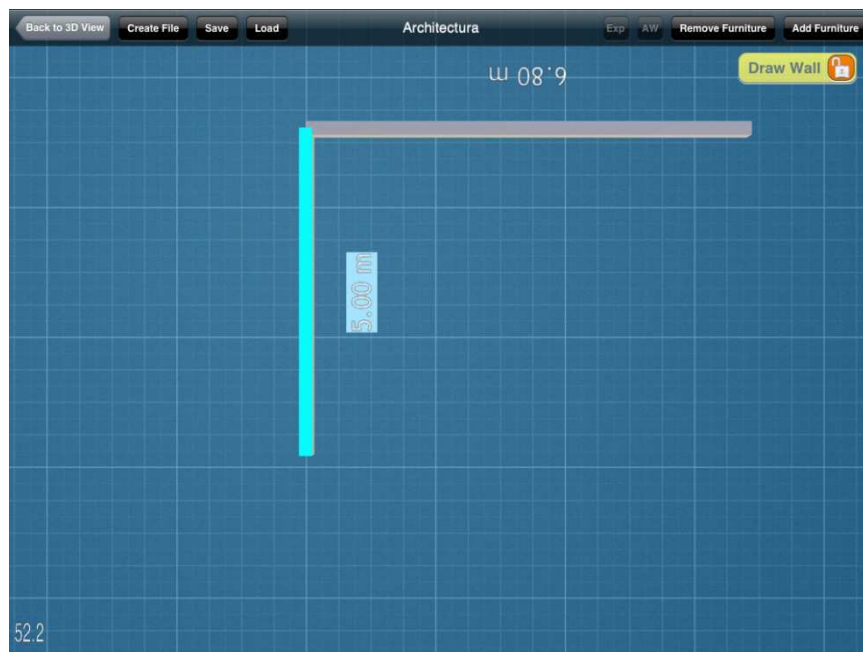


Figure 9-4

The room gets filled in with floor after it is closed (Figure 9-5).

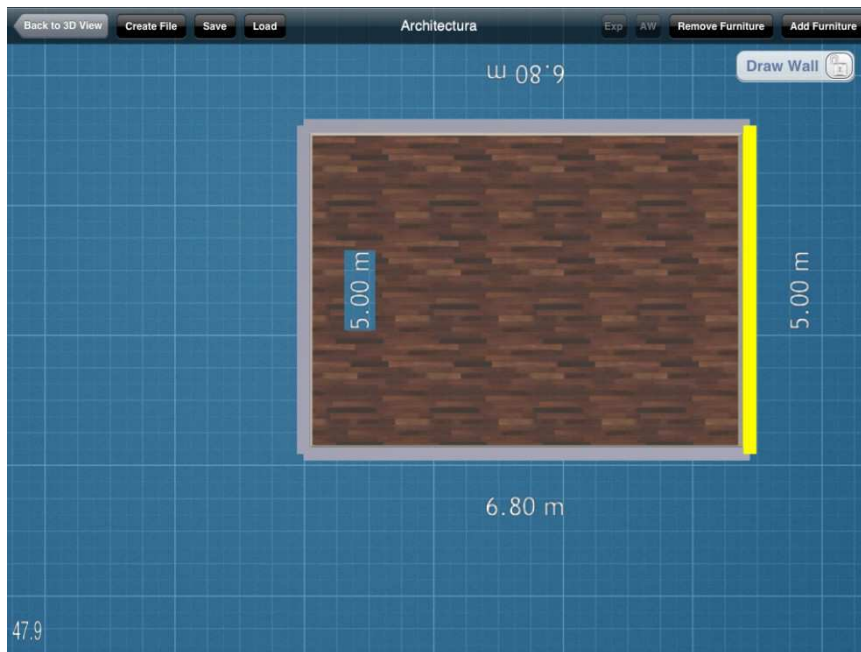


Figure 9-5

Adding a second room; the walls automatically split, when a new wall is dragged out of an existing one. The user is assisted by the application to stay inside of an existing wall. (Figure 9-6)



Figure 9-6

Adding a third room (Figure 9-7).



Figure 9-7

Moving back to the “3D view” (Figure 9-8).

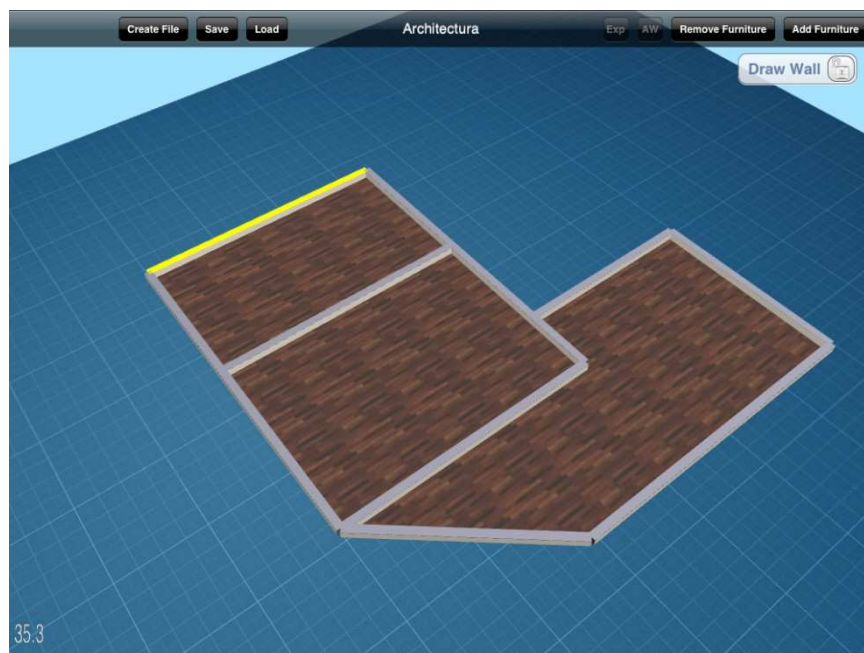


Figure 9-8

Selecting a wall by tapping; the wall is highlighted, action arrows are shown for moving and resizing the wall (Figure 9-9).

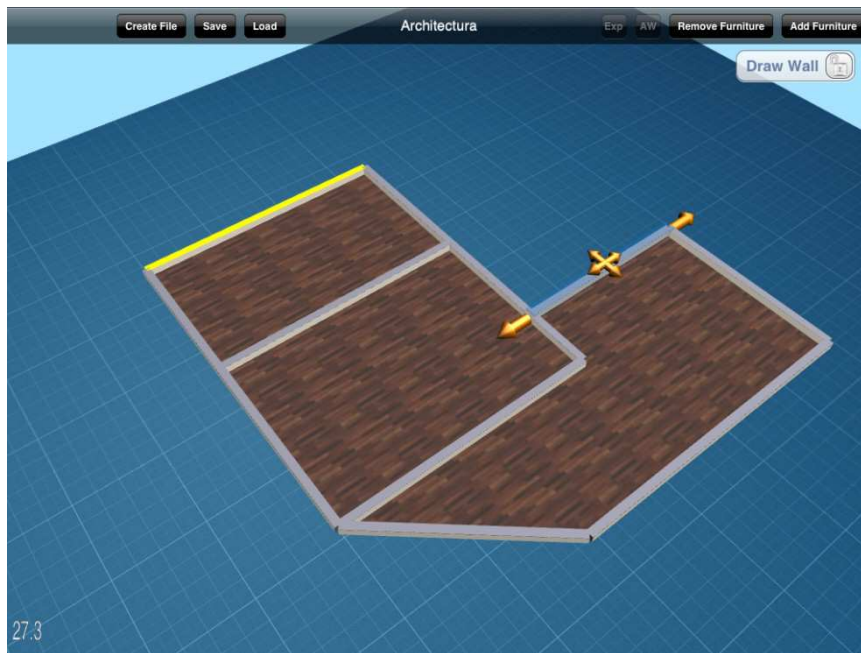


Figure 9-9

A second tap opens a context menu for the wall. The height and thickness of the wall can be adjusted here. The wall can be removed from this menu, or extended by adding windows and doors. (Figure 9-10)

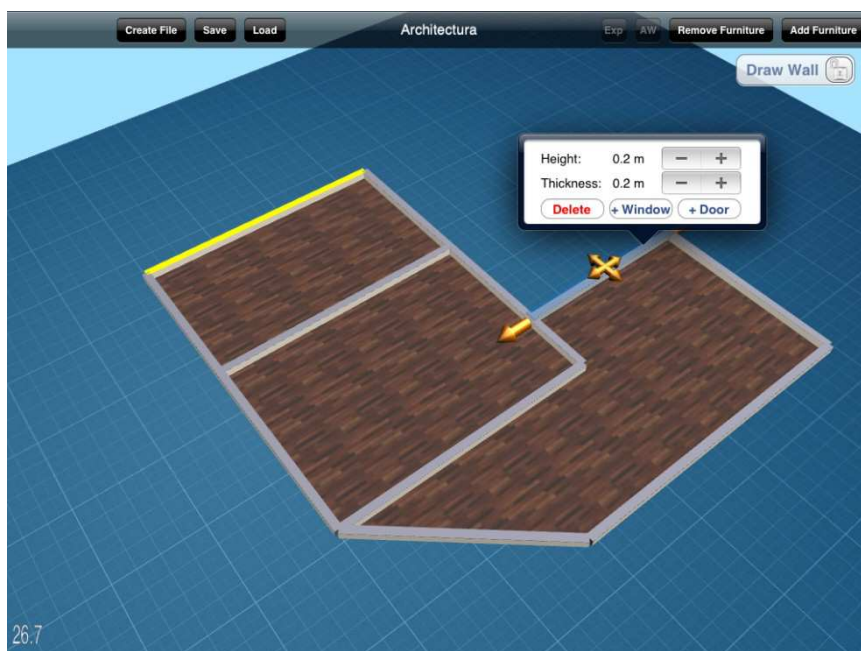


Figure 9-10

The height of two walls has been adjusted (Figure 9-11).

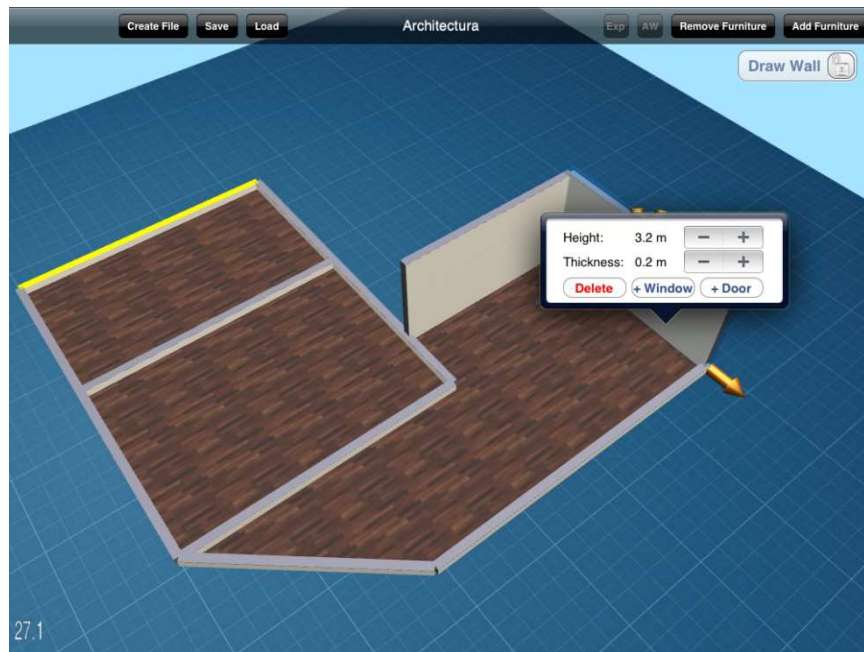


Figure 9-11

Two windows added to the wall (Figure 9-12).

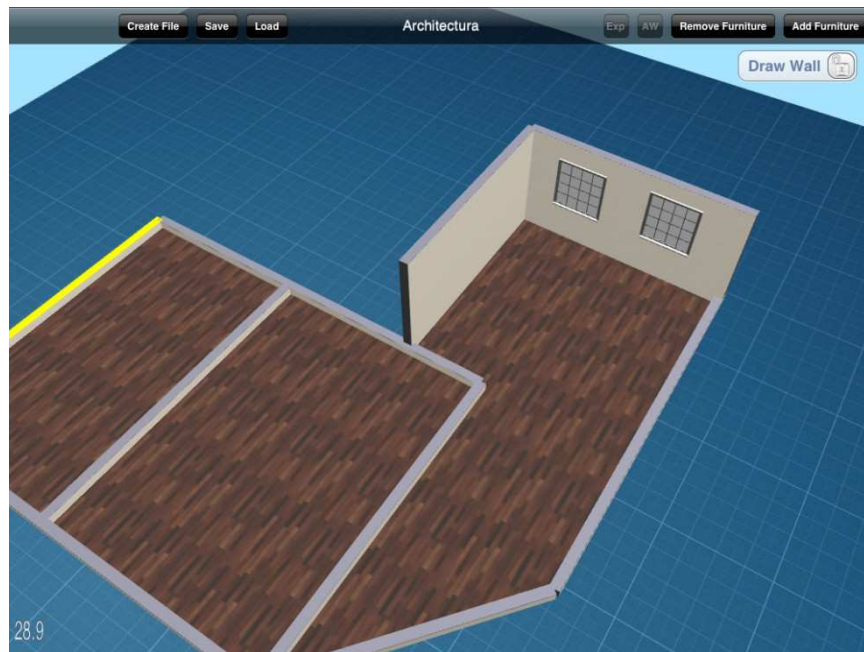


Figure 9-12

Furniture items added to the scene (Figure 9-13).



Figure 9-13

The chair can not be placed inside a wall. A collision is detected. (Figure 9-14)

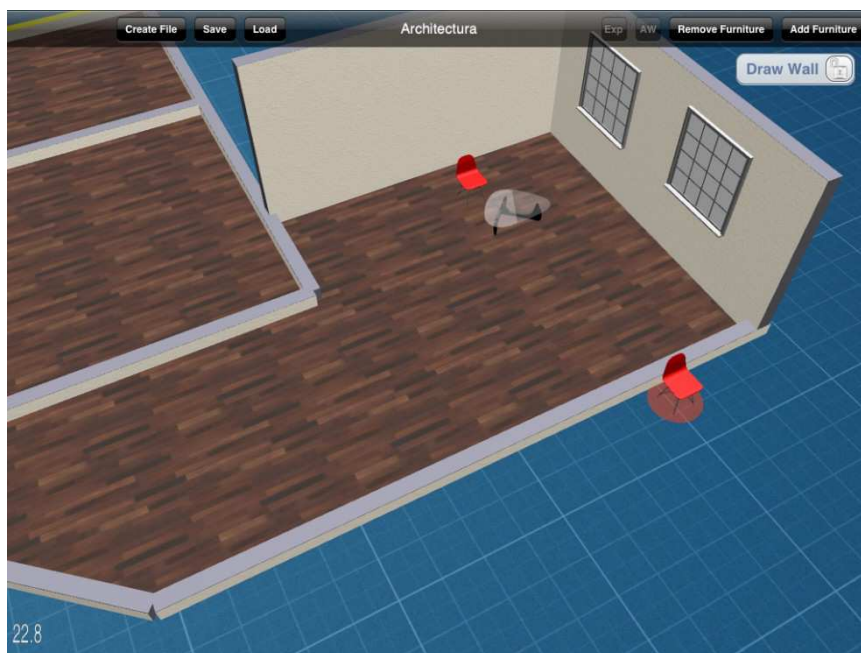


Figure 9-14

Furniture can be moved around and rotated. Walls can be resized and moved around.
(Figure 9-15)

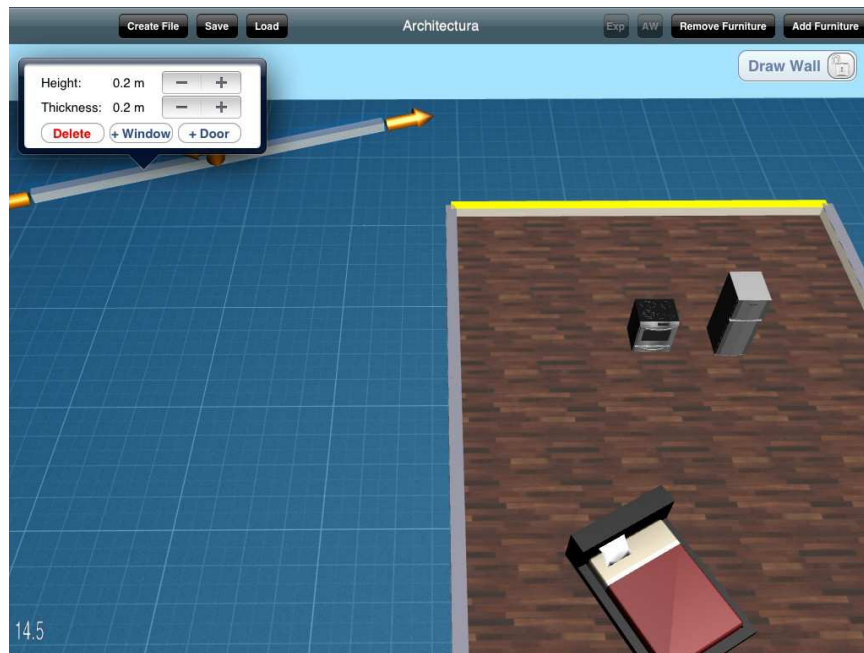


Figure 9-15

10 CD content

- src/
 - Objective-C code
 - XCode project
 - Ruby code
 - Clojure code
- thesis/
 - pdf version of the thesis

The gesture drawings (Table 3-1) are licensed under the [Creative Commons Attribution-Share Alike 3.0 Unported](#) license. They were published by Wikipedia user [GRPH3B18](#).